

# Babel

Code

Version 25.6  
2025/03/27

Javier Bezos  
Current maintainer

Johannes L. Braams  
Original author

Localization and  
internationalization

Unicode

TeX

LuaTeX

pdfTeX

XeTeX

# Contents

<b>1</b>	<b>Identification and loading of required files</b>	<b>3</b>
<b>2</b>	<b>locale directory</b>	<b>3</b>
<b>3</b>	<b>Tools</b>	<b>3</b>
3.1	A few core definitions . . . . .	8
3.2	TeX: babel.sty (start) . . . . .	8
3.3	base . . . . .	9
3.4	key=value options and other general option . . . . .	10
3.5	Post-process some options . . . . .	11
3.6	Plain: babel.def (start) . . . . .	13
<b>4</b>	<b>babel.sty and babel.def (common)</b>	<b>13</b>
4.1	Selecting the language . . . . .	15
4.2	Errors . . . . .	23
4.3	More on selection . . . . .	24
4.4	Short tags . . . . .	25
4.5	Compatibility with language.def . . . . .	25
4.6	Hooks . . . . .	26
4.7	Setting up language files . . . . .	27
4.8	Shorthands . . . . .	29
4.9	Language attributes . . . . .	38
4.10	Support for saving and redefining macros . . . . .	39
4.11	French spacing . . . . .	40
4.12	Hyphens . . . . .	41
4.13	Multiencoding strings . . . . .	43
4.14	Tailor captions . . . . .	48
4.15	Making glyphs available . . . . .	49
4.15.1	Quotation marks . . . . .	49
4.15.2	Letters . . . . .	50
4.15.3	Shorthands for quotation marks . . . . .	51
4.15.4	Umlauts and tremas . . . . .	52
4.16	Layout . . . . .	53
4.17	Load engine specific macros . . . . .	54
4.18	Creating and modifying languages . . . . .	54
4.19	Main loop in ‘provide’ . . . . .	61
4.20	Processing keys in ini . . . . .	65
4.21	French spacing (again) . . . . .	70
4.22	Handle language system . . . . .	72
4.23	Numerals . . . . .	73
4.24	Casing . . . . .	74
4.25	Getting info . . . . .	75
4.26	BCP 47 related commands . . . . .	76
<b>5</b>	<b>Adjusting the Babel behavior</b>	<b>77</b>
5.1	Cross referencing macros . . . . .	79
5.2	Layout . . . . .	82
5.3	Marks . . . . .	82
5.4	Other packages . . . . .	83
5.4.1	ifthen . . . . .	83
5.4.2	varioref . . . . .	84
5.4.3	hhline . . . . .	84
5.5	Encoding and fonts . . . . .	85
5.6	Basic bidi support . . . . .	86
5.7	Local Language Configuration . . . . .	90
5.8	Language options . . . . .	90

<b>6</b>	<b>The kernel of Babel</b>	<b>94</b>
<b>7</b>	<b>Error messages</b>	<b>94</b>
<b>8</b>	<b>Loading hyphenation patterns</b>	<b>97</b>
<b>9</b>	<b>luatex + xetex: common stuff</b>	<b>101</b>
<b>10</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>105</b>
10.1	XeTeX . . . . .	105
10.2	Support for interchar . . . . .	107
10.3	Layout . . . . .	109
10.4	8-bit TeX . . . . .	110
10.5	LuaTeX . . . . .	111
10.6	Southeast Asian scripts . . . . .	117
10.7	CJK line breaking . . . . .	119
10.8	Arabic justification . . . . .	121
10.9	Common stuff . . . . .	125
10.10	Automatic fonts and ids switching . . . . .	125
10.11	Bidi . . . . .	132
10.12	Layout . . . . .	134
10.13	Lua: transforms . . . . .	144
10.14	Lua: Auto bidi with basic and basic-r . . . . .	153
<b>11</b>	<b>Data for CJK</b>	<b>165</b>
<b>12</b>	<b>The ‘nil’ language</b>	<b>165</b>
<b>13</b>	<b>Calendars</b>	<b>166</b>
13.1	Islamic . . . . .	166
13.2	Hebrew . . . . .	168
13.3	Persian . . . . .	172
13.4	Coptic and Ethiopic . . . . .	173
13.5	Buddhist . . . . .	173
<b>14</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>174</b>
14.1	Not renaming hyphen.tex . . . . .	174
14.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	175
14.3	General tools . . . . .	176
14.4	Encoding related macros . . . . .	179
<b>15</b>	<b>Acknowledgements</b>	<b>182</b>

The babel package 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 in real documents only as documented (except, of course, if you want to explore and test them).

## 1. Identification and loading of required files

The babel package after unpacking consists of the following files:

**babel.sty** is the  $\LaTeX$  package, which set options and load language styles.

**babel.def** is loaded by Plain.

**switch.def** defines macros to set and switch languages (it loads part babel.def).

**plain.def** is not used, and just loads babel.def, for compatibility.

**hyphen.cfg** is the file to be used when generating the formats to load hyphenation patterns.

There some additional tex, def and lua files.

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with `<@name@>` at the appropriate places in the source code and defined with either `<<name=value>>`, or with a series of lines between `<<*name>>` and `<</name>>`. The latter is cumulative (e.g., with *More package options*). That brings a little bit of literate programming. The guards `<-name>` and `<+name>` have been redefined, too. See `babel.ins` for further details.

## 2. locale directory

A required component of babel is a set of ini files with basic definitions for about 300 languages. They are distributed as a separate zip file, not packed as dtx. Many 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 (e.g., there are no geographic areas in Spanish). Not all include LICR variants.

babel-\*.ini files contain the actual data; babel-\*.tex files are basically proxies to the corresponding ini files.

See [Keys in ini files](#) in the the babel site.

## 3. Tools

```
1 <<version=25.6>>
2 <<date=2025/03/27>>
```

**Do not use the following macros in ldf files. They may change in the future.** This applies mainly to those recently added for replacing, trimming and looping. The older ones, like `\bbl@afterfi`, will not change. We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in  $\LaTeX$  is executed twice, but we need them when defining options and `babel.def` cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1{#2}}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@carg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
16 \def\bbl@cs#1{\csname bbl@#1\endcsname}
17 \def\bbl@cl#1{\csname bbl@#1@languagenamename\endcsname}
18 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
19 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
```

```

20 \def\bbl@loop#1#2#3,{%
21   \ifx\@nnil#3\relax\else
22     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
23   \fi}
24 \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.

```

25 \def\bbl@add@list#1#2{%
26   \edef#1{%
27     \bbl@ifunset{\bbl@stripslash#1}%
28     }%
29     {\ifx#1@empty\else#1,\fi}%
30   #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>1</sup>. These macros will break if another `\if... \fi` statement appears in one of the arguments and it is not enclosed in braces.

```

31 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
32 \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`, `\langle` for `\noexpand` applied to a built macro name (which does not define the macro if undefined to `\relax`, because it is created locally), and `\[. . .]` for one-level expansion (where `. . .` is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```

33 \def\bbl@exp#1{%
34   \begingroup
35   \let\<\noexpand
36   \let\<\bbl@exp@en
37   \let\[\bbl@exp@ue
38   \edef\bbl@exp@aux{\endgroup#1}%
39   \bbl@exp@aux}
40 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
41 \def\bbl@exp@ue#1]{%
42   \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.

```

43 \def\bbl@tempa#1{%
44   \long\def\bbl@trim##1##2{%
45     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46   \def\bbl@trim@c{%
47     \ifx\bbl@trim@a\@sptoken
48       \expandafter\bbl@trim@b
49     \else
50       \expandafter\bbl@trim@b\expandafter#1%
51     \fi}%
52   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
53 \bbl@tempa{ }
54 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bbl@trim@def#1{\bbl@trim{def#1}}

```

<sup>1</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

**\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 `\ifcurname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcurname` being implicitly set to `\relax` by the `\curname` test.

```

56 \begingroup
57 \gdef\bbl@ifunset#1{%
58   \expandafter\ifx\curname#1\endcurname\relax
59   \expandafter\@firstoftwo
60   \else
61   \expandafter\@secondoftwo
62   \fi}
63 \bbl@ifunset{ifcurname}%
64 {}%
65 {\gdef\bbl@ifunset#1{%
66   \ifcurname#1\endcurname
67   \expandafter\ifx\curname#1\endcurname\relax
68   \bbl@afterelse\expandafter\@firstoftwo
69   \else
70   \bbl@afterfi\expandafter\@secondoftwo
71   \fi
72   \else
73   \expandafter\@firstoftwo
74   \fi}}
75 \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, i.e., not `\relax` and not empty,

```

76 \def\bbl@ifblank#1{%
77   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil#4}
79 \def\bbl@ifset#1#2#3{%
80   \bbl@ifunset{#1}{#3}{\bbl@exp{\@nil\bbl@ifblank{\@nameuse{#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` (i.e., the macro thus named, not an empty argument, which is what you get with `<key>=` and no value).

```

81 \def\bbl@forkv#1#2{%
82   \def\bbl@kvcmd##1##2##3{#2}%
83   \bbl@kvnext#1,\@nil,}
84 \def\bbl@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86   \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
87   \expandafter\bbl@kvnext
88   \fi}
89 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
90   \bbl@trim\def\bbl@forkv@a{#1}%
91   \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).

```

92 \def\bbl@vforeach#1#2{%
93   \def\bbl@forcmd##1{#2}%
94   \bbl@fornext#1,\@nil,}
95 \def\bbl@fornext#1,{%
96   \ifx\@nil#1\relax\else
97   \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
98   \expandafter\bbl@fornext
99   \fi}
100 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

Some code should be executed once. The first argument is a flag.

```

101 \global\let\bbl@done\@empty

```

```

102 \def\bbl@once#1#2{%
103 \bbl@xin@{,#1,}{,\bbl@done,}%
104 \ifin@\else
105 #2%
106 \xdef\bbl@done{\bbl@done,#1,}%
107 \fi}
108% \end{macrode}
109%
110% \macro{\bbl@replace}
111%
112% Returns implicitly |\toks@| with the modified string.
113%
114% \begin{macrocode}
115 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
116 \toks@{}}%
117 \def\bbl@replace@aux##1#2##2#2{%
118 \ifx\bbl@nil##2%
119 \toks@\expandafter{\the\toks@##1}%
120 \else
121 \toks@\expandafter{\the\toks@##1#3}%
122 \bbl@afterfi
123 \bbl@replace@aux##2#2%
124 \fi}%
125 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
126 \edef#1{\the\toks@}}

```

An extension to the previous macro. It takes into account the parameters, and it is string based (i.e., 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 checking the replacement is really necessary or just paranoia).

```

127 \ifx\detokenize@undefined\else % Unused macros if old Plain TeX
128 \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
129 \def\bbl@tempa{#1}%
130 \def\bbl@tempb{#2}%
131 \def\bbl@tempe{#3}}
132 \def\bbl@sreplace#1#2#3{%
133 \begingroup
134 \expandafter\bbl@parsedef\meaning#1\relax
135 \def\bbl@tempc{#2}%
136 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
137 \def\bbl@tempd{#3}%
138 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
139 \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
140 \ifin@
141 \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
142 \def\bbl@tempc{% Expanded an executed below as 'uplevel'
143 \\makeatletter % "internal" macros with @ are assumed
144 \\scantokens{%
145 \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}%
146 \noexpand\noexpand}%
147 \catcode64=\the\catcode64\relax}% Restore @
148 \else
149 \let\bbl@tempc@empty % Not \relax
150 \fi
151 \bbl@exp{% For the 'uplevel' assignments
152 \endgroup
153 \bbl@tempc}} % empty or expand to set #1 with changes
154 \fi

```

Two further tools. \bbl@ifsamestring 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 pdfL<sub>ATE</sub>X, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

155 \def\bb@ifsamestring#1#2{%
156   \begingroup
157     \protected@edef\bb@tempb{#1}%
158     \edef\bb@tempb{\expandafter\strip@prefix\meaning\bb@tempb}%
159   \protected@edef\bb@tempc{#2}%
160   \edef\bb@tempc{\expandafter\strip@prefix\meaning\bb@tempc}%
161   \ifx\bb@tempb\bb@tempc
162     \aftergroup\@firstoftwo
163   \else
164     \aftergroup\@secondoftwo
165   \fi
166 \endgroup}
167 \chardef\bb@engine=%
168 \ifx\directlua\@undefined
169   \ifx\XeTeXinputencoding\@undefined
170     \z@
171   \else
172     \tw@
173   \fi
174 \else
175   \@ne
176 \fi

```

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

```

177 \def\bb@bsphack{%
178   \ifhmode
179     \hskip\z@skip
180     \def\bb@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
181   \else
182     \let\bb@esphack\@empty
183   \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`.

```

184 \def\bb@cased{%
185   \ifx\oe\OE
186     \expandafter\in@\expandafter
187     {\expandafter\OE\expandafter}\expandafter{\oe}%
188   \ifin@
189     \bb@afterelse\expandafter\MakeUppercase
190   \else
191     \bb@afterfi\expandafter\MakeLowercase
192   \fi
193 \else
194   \expandafter\@firstofone
195 \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`).

```

196 \def\bb@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
197   \toks@\expandafter\expandafter\expandafter{%
198     \csname extras\language\endcsname}%
199   \bb@exp{\in@{#1}}{\the\toks@}}%
200   \ifin@\else
201     \@temptokena{#2}%
202     \edef\bb@tempc{\the\@temptokena\the\toks@}%
203     \toks@\expandafter{\bb@tempc#3}%
204     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
205   \fi}
206 <</Basic macros>>

```

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



```

207 <<{*Make sure ProvidesFile is defined}>> ≡
208 \ifx\ProvidesFile\undefined
209   \def\ProvidesFile#1[#2 #3 #4]{%
210     \wlog{File: #1 #4 #3 <#2>}%
211     \let\ProvidesFile\undefined}
212 \fi
213 <</Make sure ProvidesFile is defined>>

```

### 3.1. A few core definitions

**\language** Just for compatibility, for not to touch `hyphen.cfg`.

```

214 <<{*Define core switching macros}>> ≡
215 \ifx\language\undefined
216   \csname newcount\endcsname\language
217 \fi
218 <</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.

```

219 <<{*Define core switching macros}>> ≡
220 \countdef\last@language=19
221 \def\addlanguage{\csname newlanguage\endcsname}
222 <</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.

### 3.2. $\LaTeX$ : `babel.sty` (start)

Here starts the style file for  $\LaTeX$ . It also takes care of a number of compatibility issues with other packages.

```

223 <{*package}>
224 \NeedsTeXFormat{LaTeX2e}
225 \ProvidesPackage{babel}%
226 [ <@date@> v<@version@> %%NB%%
227 The multilingual framework for pdfLaTeX, LuaLaTeX and XeLaTeX]

```

Start with some “private” debugging tools, and then define macros for errors. The global lua ‘space’ Babel is declared here, too (inside the test for debug).

```

228 \ifpackagewith{babel}{debug}
229   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
230   \let\bbl@debug\@firstofone
231   \ifx\directlua\undefined\else
232     \directlua{
233       Babel = Babel or {}
234       Babel.debug = true }%
235     \input{babel-debug.tex}%
236   \fi
237   {\providecommand\bbl@trace[1]}%
238   \let\bbl@debug\@gobble
239   \ifx\directlua\undefined\else
240     \directlua{
241       Babel = Babel or {}
242       Babel.debug = false }%
243   \fi}

```

Macros to deal with errors, warnings, etc. Errors are stored in a separate file.

```

244 \def\bb@error#1{% Implicit #2#3#4
245   \begingroup
246     \catcode`\=0 \catcode`\==12 \catcode`\`=12
247     \input errbabel.def
248   \endgroup
249   \bb@error{#1}}
250 \def\bb@warning#1{%
251   \begingroup
252     \def\{\MessageBreak}%
253     \PackageWarning{babel}{#1}%
254   \endgroup}
255 \def\bb@infowarn#1{%
256   \begingroup
257     \def\{\MessageBreak}%
258     \PackageNote{babel}{#1}%
259   \endgroup}
260 \def\bb@info#1{%
261   \begingroup
262     \def\{\MessageBreak}%
263     \PackageInfo{babel}{#1}%
264   \endgroup}

```

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.

```

265 <@Basic macros@>
266 \ifpackagewith{babel}{silent}
267   {\let\bb@info@gobble
268    \let\bb@infowarn@gobble
269    \let\bb@warning@gobble}
270 {}
271 %
272 \def\AfterBabelLanguage#1{%
273   \global\expandafter\bb@add\csname#1.ldf-h@k\endcsname}%

```

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

```

274 \ifx\bb@languages\undefined\else
275   \begingroup
276     \catcode`\^^I=12
277     \@ifpackagewith{babel}{showlanguages}{%
278       \begingroup
279         \def\bb@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
280         \wlog{<*languages>}%
281         \bb@languages
282         \wlog{</languages>}%
283       \endgroup}{%
284     \endgroup
285     \def\bb@elt#1#2#3#4{%
286       \ifnum#2=\z@
287         \gdef\bb@nulllanguage{#1}%
288         \def\bb@elt##1##2##3##4{%
289           \fi}%
290     \bb@languages
291 \fi%

```

### 3.3. base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that L<sup>A</sup>T<sub>E</sub>X 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.

```

292 \bbl@trace{Defining option 'base'}
293 \@ifpackagewith{babel}{base}{%
294   \let\bbl@onlyswitch\@empty
295   \let\bbl@provide@locale\relax
296   \input babel.def
297   \let\bbl@onlyswitch\@undefined
298   \ifx\directlua\@undefined
299     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
300   \else
301     \input luababel.def
302     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
303   \fi
304   \DeclareOption{base}{}%
305   \DeclareOption{showlanguages}{}%
306   \ProcessOptions
307   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
308   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
309   \global\let\@ifl@ter@\@ifl@ter
310   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
311   \endinput}{}%

```

### 3.4. 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`.

```

312 \bbl@trace{key=value and another general options}
313 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
314 \def\bbl@tempb#1.#2{% Remove trailing dot
315   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
316 \def\bbl@tempe#1=#2\@@{%
317   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
318 \def\bbl@tempd#1.#2\@nnil{%%^A TODO. Refactor lists?
319   \ifx\@empty#2%
320     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
321   \else
322     \in@{,provide=}{,#1}%
323     \ifin@
324       \edef\bbl@tempc{%
325         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
326     \else
327       \in@{ $modifiers$ }{ $#1$ }%%^A TODO. Allow spaces.
328       \ifin@
329         \bbl@tempe#2\@@
330       \else
331         \in@{=}{#1}%
332         \ifin@
333           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
334         \else
335           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
336           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
337         \fi
338       \fi
339     \fi
340   \fi}
341 \let\bbl@tempc\@empty
342 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
343 \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.

```

344 \DeclareOption{KeepShorthandsActive}{}
345 \DeclareOption{activeacute}{}
346 \DeclareOption{activegrave}{}
347 \DeclareOption{debug}{}
348 \DeclareOption{noconfigs}{}
349 \DeclareOption{showlanguages}{}
350 \DeclareOption{silent}{}
351 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
352 \chardef\bbl@iniflag\z@
353 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
354 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
355 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
356 % Don't use. Experimental. TODO.
357 \newif\ifbbl@single
358 \DeclareOption{selectors=off}{\bbl@singletrue}
359 <@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  $\langle key \rangle = \langle value \rangle$ , 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.

```

360 \let\bbl@opt@shorthands\@nnil
361 \let\bbl@opt@config\@nnil
362 \let\bbl@opt@main\@nnil
363 \let\bbl@opt@headfoot\@nnil
364 \let\bbl@opt@layout\@nnil
365 \let\bbl@opt@provide\@nnil

```

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

```

366 \def\bbl@tempa#1=#2\bbl@tempa{%
367   \bbl@csarg\ifx{opt@#1}\@nnil
368   \bbl@csarg\edef{opt@#1}{#2}%
369   \else
370   \bbl@error{bad-package-option}{#1}{#2}{}%
371   \fi}

```

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

```

372 \let\bbl@language@opts\@empty
373 \DeclareOption*{%
374   \bbl@xin@{\string=}{\CurrentOption}%
375   \ifin@
376   \expandafter\bbl@tempa\CurrentOption\bbl@tempa
377   \else
378   \bbl@add@list\bbl@language@opts{\CurrentOption}%
379   \fi}

```

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

```

380 \ProcessOptions*

```

### 3.5. Post-process some options

```

381 \ifx\bbl@opt@provide\@nnil
382   \let\bbl@opt@provide\@empty % %%% MOVE above
383 \else
384   \chardef\bbl@iniflag\@ne
385   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
386     \in@{,provide,}{, #1,}%
387     \ifin@
388     \def\bbl@opt@provide{#2}%
389     \fi}

```

390 \fi

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=...`

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

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

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

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

```
412 \edef\bbl@opt@shorthands{%
413   \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

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

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

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

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

For the option `safe` we use a different approach – `\bbl@opt@safe` says which macros are redefined (B for bibs and R for refs). By default, both are currently set, but in a future release it will be set to none.

```
425 \ifx\bbl@opt@safe\@undefined
426   \def\bbl@opt@safe{BR}
427   % \let\bbl@opt@safe\@empty % Pending of \cite
428 \fi
```

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

Optimization: if there is no layout, just do nothing.

```
429 \bbl@trace{Defining IfBabelLayout}
430 \ifx\bbl@opt@layout\@nnil
431   \newcommand\IfBabelLayout[3]{#3}%
432 \else
433   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
```

```

434 \in@{,layout,}{, #1,}%
435 \ifin@
436 \def\bbl@opt@layout{#2}%
437 \bbl@replace\bbl@opt@layout{ }{.}%
438 \fi}
439 \newcommand\IfBabelLayout[1]{%
440 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
441 \ifin@
442 \expandafter\@firstoftwo
443 \else
444 \expandafter\@secondoftwo
445 \fi}
446 \fi
447 \</package>

```

### 3.6. Plain: babel.def (start)

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*.

First, exit immediately if previously loaded.

```

448 \<core>
449 \ifx\ldf@quit\undefined\else
450 \endinput\fi % Same line!
451 \<@Make sure ProvidesFile is defined>
452 \ProvidesFile{babel.def}[<@date> v<@version> Babel common definitions]
453 \ifx\AtBeginDocument\undefined %^^A TODO. change test.
454 \<@Emulate LaTeX>
455 \fi
456 \<@Basic macros>
457 \</core>

```

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

## 4. babel.sty and babel.def (common)

```

458 \<package | core>
459 \def\bbl@version{<@version>}
460 \def\bbl@date{<@date>}
461 \<@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.

```

462 \def\adddialect#1#2{%
463 \global\chardef#1#2\relax
464 \bbl@usehooks{adddialect}{#1}{#2}%
465 \begingroup
466 \count@#1\relax
467 \def\bbl@elt##1##2###3###4{%
468 \ifnum\count@=#2\relax
469 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
470 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
471 set to \expandafter\string\csname l@##1\endcsname\%
472 (\string\language\the\count@). Reported}%
473 \def\bbl@elt####1####2####3####4{%
474 \fi}%
475 \bbl@cs{languages}%
476 \endgroup}

```

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

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

```

477 \def\bbl@fixname#1{%
478   \begingroup
479   \def\bbl@tempe{l@}%
480   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
481   \bbl@tempd
482     {\lowercase\expandafter{\bbl@tempd}%
483      {\uppercase\expandafter{\bbl@tempd}%
484       \@empty
485        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
486         \uppercase\expandafter{\bbl@tempd}}}%
487       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
488        \lowercase\expandafter{\bbl@tempd}}}%
489   \@empty
490   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
491   \bbl@tempd
492   \bbl@exp{\@bbl@usehooks{language}{\language}{#1}}
493 \def\bbl@iflanguage#1{%
494   \@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 BCP 47 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`.

```

495 \def\bbl@bcpcase#1#2#3#4\@#5{%
496   \ifx\@empty#3%
497     \uppercase{\def#5{#1#2}}%
498   \else
499     \uppercase{\def#5{#1}}%
500     \lowercase{\edef#5{#5#2#3#4}}%
501   \fi}
502 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
503   \let\bbl@bcp\relax
504   \lowercase{\def\bbl@tempa{#1}}%
505   \ifx\@empty#2%
506     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
507   \else\ifx\@empty#3%
508     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
509     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
510     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
511     {}%
512     \ifx\bbl@bcp\relax
513       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
514     \fi
515   \else
516     \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
517     \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
518     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
519     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
520     {}%
521     \ifx\bbl@bcp\relax
522       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
523       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
524       {}%
525     \fi
526     \ifx\bbl@bcp\relax
527       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
528       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
529       {}%
530     \fi

```

```

531 \ifx\bbl@bcp\relax
532 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
533 \fi
534 \fi\fi}
535 \let\bbl@initoload\relax

```

**\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.

```

536 \def\iflanguage#1{%
537 \bbl@iflanguage{#1}{%
538 \ifnum\csname l@#1\endcsname=\language
539 \expandafter\@firstoftwo
540 \else
541 \expandafter\@secondoftwo
542 \fi}}

```

## 4.1. Selecting the language

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

```

543 \let\bbl@select@type\z@
544 \edef\selectlanguage{%
545 \noexpand\protect
546 \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`.

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

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

```
548 \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.

```
549 \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**



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

```

550 \def\bbl@push@language{%
551   \ifx\languagename\undefined\else
552     \ifx\currentgrouplevel\undefined
553       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
554     \else
555       \ifnum\currentgrouplevel=\z@
556         \xdef\bbl@language@stack{\languagename+}%
557       \else
558         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
559       \fi
560     \fi
561 \fi}

```

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

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

```

562 \def\bbl@pop@lang#1+#2\@@{%
563   \edef\languagename{#1}%
564   \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).

```

565 \let\bbl@ifrestoring\@secondoftwo
566 \def\bbl@pop@language{%
567   \expandafter\bbl@pop@lang\bbl@language@stack\@@
568   \let\bbl@ifrestoring\@firstoftwo
569   \expandafter\bbl@set@language\expandafter{\languagename}%
570   \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).

```

571 \chardef\localeid\z@
572 \def\bbl@id@last{0} % No real need for a new counter
573 \def\bbl@id@assign{%
574   \bbl@ifunset{bbl@id@\languagename}%
575     {\count@\bbl@id@last\relax
576     \advance\count@\@ne
577     \global\bbl@csarg\chardef{id@\languagename}\count@
578     \edef\bbl@id@last{\the\count@}%
579     \ifcase\bbl@engine\or
580       \directlua{
581         Babel.locale_props[\bbl@id@last] = {}
582         Babel.locale_props[\bbl@id@last].name = '\languagename'
583         Babel.locale_props[\bbl@id@last].vars = {}
584       }%
585     \fi}%
586   {}%
587   \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of \selectlanguage. In case it is used as environment, declare \endselectlanguage, just for safety.

```

588 \expandafter\def\csname selectlanguage \endcsname#1{%

```

```

589 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
590 \bbl@push@language
591 \aftergroup\bbl@pop@language
592 \bbl@set@language{#1}}
593 \let\endselectlanguage\relax

```

**\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 `\languagename` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards.

We also write a command to change the current language in the auxiliary files.

`\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).

```

594 \def\BabelContentsFiles{toc,lof,lot}
595 \def\bbl@set@language#1{% from selectlanguage, pop@
596 % The old buggy way. Preserved for compatibility, but simplified
597 \edef\languagename{\expandafter\string#1\@empty}%
598 \select@language{\languagename}%
599 % write to auxs
600 \expandafter\ifx\cscname date\languagename\endcscname\relax\else
601 \if@filesw
602 \ifx\babel@aux@\gobbletwo\else % Set if single in the first, redundant
603 \bbl@savelastskip
604 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname{}}}%
605 \bbl@restorelastskip
606 \fi
607 \bbl@usehooks{write}{}%
608 \fi
609 \fi}
610 %
611 \let\bbl@restorelastskip\relax
612 \let\bbl@savelastskip\relax
613 %
614 \def\select@language#1{% from set@, babel@aux, babel@toc
615 \ifx\bbl@selectorname\@empty
616 \def\bbl@selectorname{select}%
617 \fi
618 % set hmap
619 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
620 % set name (when coming from babel@aux)
621 \edef\languagename{#1}%
622 \bbl@fixname\languagename
623 % define \localename when coming from set@, with a trick
624 \ifx\scantokens\undefined
625 \def\localename{??}%
626 \else
627 \bbl@exp{\scantokens{\def\localename{\languagename}\noexpand}\relax}%
628 \fi
629 %^^A TODO. name@map must be here?
630 \bbl@provide@locale
631 \bbl@iflanguage\languagename{%
632 \let\bbl@select@type\z@
633 \expandafter\bbl@switch\expandafter{\languagename}}
634 \def\babel@aux#1#2{%
635 \select@language{#1}%
636 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
637 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}%^^A TODO - plain?
638 \def\babel@toc#1#2{%

```

639 `\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{language}` 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 `\select language`, 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 `\{language\}hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\{language\}hyphenmins` will be used.

No text is supposed to be added with switching captions and date, so we remove any spurious spaces with `\bbl@bsphack` and `\bbl@esphack`.

```
640 \newif\ifbbl@usedategroup
641 \let\bbl@savextras\@empty
642 \def\bbl@switch#1{% from select@, foreign@
643 % make sure there is info for the language if so requested
644 \bbl@ensureinfo{#1}%
645 % restore
646 \originalTeX
647 \expandafter\def\expandafter\originalTeX\expandafter{%
648 \csname noextras#1\endcsname
649 \let\originalTeX\@empty
650 \babel@beginsave}%
651 \bbl@usehooks{afterreset}{}%
652 \languageshorthands{none}%
653 % set the locale id
654 \bbl@id@assign
655 % switch captions, date
656 \bbl@bsphack
657 \ifcase\bbl@select@type
658 \csname captions#1\endcsname\relax
659 \csname date#1\endcsname\relax
660 \else
661 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
662 \ifin@
663 \csname captions#1\endcsname\relax
664 \fi
665 \bbl@xin@{,date,}{,\bbl@select@opts,}%
666 \ifin@ % if \foreign... within \<language>date
667 \csname date#1\endcsname\relax
668 \fi
669 \fi
670 \bbl@esphack
671 % switch extras
672 \csname bbl@preextras@#1\endcsname
673 \bbl@usehooks{beforeextras}{}%
674 \csname extras#1\endcsname\relax
675 \bbl@usehooks{afterextras}{}%
676 % > babel-ensure
677 % > babel-sh-<short>
678 % > babel-bidi
679 % > babel-fontspec
680 \let\bbl@savextras\@empty
681 % hyphenation - case mapping
682 \ifcase\bbl@opt@hyphenmap\or
683 \def\BabelLower##1##2{\lccode##1=##2\relax}%
684 \ifnum\bbl@hymapsel>4\else
685 \csname\language @bbl@hyphenmap\endcsname
686 \fi
```

```

687 \chardef\bblopt@hyphenmap\z@
688 \else
689 \ifnum\bblopt@hymapsel>\bblopt@hyphenmap\else
690 \csname\language @bblopt@hyphenmap\endcsname
691 \fi
692 \fi
693 \let\bblopt@hymapsel\ccclv
694 % hyphenation - select rules
695 \ifnum\csname l@language\endcsname=\l@unhyphenated
696 \edef\bblopt@tempa{u}%
697 \else
698 \edef\bblopt@tempa{\bblopt@cl{\lnbrk}}%
699 \fi
700 % linebreaking - handle u, e, k (v in the future)
701 \bblopt@xin@{/u}{\bblopt@tempa}%
702 \ifin@ \else \bblopt@xin@{/e}{\bblopt@tempa} \fi % elongated forms
703 \ifin@ \else \bblopt@xin@{/k}{\bblopt@tempa} \fi % only kashida
704 \ifin@ \else \bblopt@xin@{/p}{\bblopt@tempa} \fi % padding (e.g., Tibetan)
705 \ifin@ \else \bblopt@xin@{/v}{\bblopt@tempa} \fi % variable font
706 % hyphenation - save mins
707 \babel@savevariable\lefthyphenmin
708 \babel@savevariable\righthyphenmin
709 \ifnum\bblopt@engine=\@ne
710 \babel@savevariable\hyphenationmin
711 \fi
712 \ifin@
713 % unhyphenated/kashida/elongated/padding = allow stretching
714 \language\l@unhyphenated
715 \babel@savevariable\emergencystretch
716 \emergencystretch\maxdimen
717 \babel@savevariable\hbadness
718 \hbadness\@M
719 \else
720 % other = select patterns
721 \bblopt@patterns{#1}%
722 \fi
723 % hyphenation - set mins
724 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
725 \set@hyphenmins\tw@\thr@@\relax
726 \@nameuse{bblopt@hyphenmins@}%
727 \else
728 \expandafter\expandafter\expandafter\set@hyphenmins
729 \csname #1hyphenmins\endcsname\relax
730 \fi
731 \@nameuse{bblopt@hyphenmins@}%
732 \@nameuse{bblopt@hyphenmins@\language}%
733 \@nameuse{bblopt@hyphenatmin@}%
734 \@nameuse{bblopt@hyphenatmin@\language}%
735 \let\bblopt@selectorname\@empty}

```

**otherlanguage** It can be used as an alternative to using the `\selectlanguage` declarative command. The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

736 \long\def\otherlanguage#1{%
737 \def\bblopt@selectorname{other}%
738 \ifnum\bblopt@hymapsel=\ccclv\let\bblopt@hymapsel\thr@@\fi
739 \csname selectlanguage \endcsname{#1}%
740 \ignorespaces}

```

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

```

741 \long\def\endotherlanguage{\@ignoretrue\ignorespaces}

```

**otherlanguage\*** It 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’. It makes use of `\foreign@language`.

```

742 \expandafter\def\csname otherlanguage*\endcsname{%
743   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
744 \def\bbl@otherlanguage@s[#1]#2{%
745   \def\bbl@selectorname{other*}%
746   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
747   \def\bbl@select@opts{#1}%
748   \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”.

```

749 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

**\foreignlanguage** 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⟨language⟩` 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.

```

750 \providecommand\bbl@beforeforeign{}
751 \edef\foreignlanguage{%
752   \noexpand\protect
753   \expandafter\noexpand\csname foreignlanguage \endcsname}
754 \expandafter\def\csname foreignlanguage \endcsname{%
755   \@ifstar\bbl@foreign@s\bbl@foreign@x}
756 \providecommand\bbl@foreign@x[3][]{%
757   \begingroup
758     \def\bbl@selectorname{foreign}%
759     \def\bbl@select@opts{#1}%
760     \let\BabelText\@firstofone
761     \bbl@beforeforeign
762     \foreign@language{#2}%
763     \bbl@usehooks{foreign}{}%
764     \BabelText{#3}% Now in horizontal mode!
765   \endgroup}
766 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
767   \begingroup
768     {\par}%
769     \def\bbl@selectorname{foreign*}%
770     \let\bbl@select@opts\@empty
771     \let\BabelText\@firstofone
772     \foreign@language{#1}%
773     \bbl@usehooks{foreign*}{}%
774     \bbl@dirparastext
775     \BabelText{#2}% Still in vertical mode!
776     {\par}%

```

```

777 \endgroup}
778 \providecommand\BabelWrapText[1]{%
779 \def\bbl@tempa{\def\BabelText###1}%
780 \expandafter\bbl@tempa\expandafter\BabelText{#1}}

```

**\foreign@language** This macro does the work for `\foreignlanguage` and the other `language*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `bbl@switch`.

```

781 \def\foreign@language#1{%
782 % set name
783 \edef\languagename{#1}%
784 \ifbbl@usedategroup
785 \bbl@add\bbl@select@opts{,date,}%
786 \bbl@usedategroupfalse
787 \fi
788 \bbl@fixname\languagename
789 \let\localename\languagename
790 % TODO. name@map here?
791 \bbl@provide@locale
792 \bbl@iflanguage\languagename{%
793 \let\bbl@select@type\@ne
794 \expandafter\bbl@switch\expandafter\languagename}}

```

The following macro executes conditionally some code based on the selector being used.

```

795 \def\IfBabelSelectorTF#1{%
796 \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
797 \ifin@
798 \expandafter\@firstoftwo
799 \else
800 \expandafter\@secondoftwo
801 \fi}

```

**\bbl@patterns** This macro selects the hyphenation patterns by changing the `\language` register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.

It also sets hyphenation exceptions, but only once, because they are global (here `language \lccode's` has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\babelhyphenation`, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that `:ENC` is taken into account) has been set, then use `\hyphenation` with both global and language exceptions and empty the latter to mark they must not be set again.

```

802 \let\bbl@hyphlist\@empty
803 \let\bbl@hyphenation@\relax
804 \let\bbl@pttnlist\@empty
805 \let\bbl@patterns@\relax
806 \let\bbl@hymapsel=\@cclv
807 \def\bbl@patterns#1{%
808 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
809 \csname l@#1\endcsname
810 \edef\bbl@tempa{#1}%
811 \else
812 \csname l@#1:\f@encoding\endcsname
813 \edef\bbl@tempa{#1:\f@encoding}%
814 \fi
815 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
816 % > luatex
817 \ifundefined{bbl@hyphenation@}{% Can be \relax!
818 \begingroup
819 \bbl@xin@{\,number\language,}{,\bbl@hyphlist}%
820 \ifin@else
821 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
822 \hyphenation{%
823 \bbl@hyphenation@

```

```

824     \ifundefined{bbl@hyphenation@#1}%
825     \@empty
826     {\space\csname bbl@hyphenation@#1\endcsname}}%
827     \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
828     \fi
829 \endgroup}}

```

**hyphenrules** It can be used to select *just* the hyphenation rules. It 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*`.

```

830 \def\hyphenrules#1{%
831   \edef\bbl@tempf{#1}%
832   \bbl@fixname\bbl@tempf
833   \bbl@iflanguage\bbl@tempf{%
834     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
835     \ifx\languageshorthands\@undefined\else
836       \languageshorthands{none}%
837     \fi
838     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
839       \set@hyphenmins\tw@\thr@@\relax
840     \else
841       \expandafter\expandafter\expandafter\set@hyphenmins
842       \csname\bbl@tempf hyphenmins\endcsname\relax
843     \fi}}
844 \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 `\(language)hyphenmins` is already defined this command has no effect.

```

845 \def\providehyphenmins#1#2{%
846   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
847   \@namedef{#1hyphenmins}{#2}%
848   \fi}

```

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

```

849 \def\set@hyphenmins#1#2{%
850   \lefthyphenmin#1\relax
851   \righthyphenmin#2\relax}

```

**\ProvidesLanguage** The identification code for each file is something that was introduced in  $\text{\LaTeX} 2_{\epsilon}$ . When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by `babel`.

Depending on the format, i.e., or if the former is defined, we use a similar definition or not.

```

852 \ifx\ProvidesFile\@undefined
853   \def\ProvidesLanguage#1[#2 #3 #4]{%
854     \wlog{Language: #1 #4 #3 <#2>}%
855   }
856 \else
857   \def\ProvidesLanguage#1{%
858     \begingroup
859     \catcode`\ 10 %
860     \@makeother\%
861     \@ifnextchar[%
862       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
863   \def\@provideslanguage#1[#2]{%
864     \wlog{Language: #1 #2}%
865     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
866     \endgroup}
867 \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`.

```
868 \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.

```
869 \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’:

```
870 \providecommand\setlocale{\bbl@error{not-yet-available}{}}{}
871 \let\uselocale\setlocale
872 \let\locale\setlocale
873 \let\selectlocale\setlocale
874 \let\textlocale\setlocale
875 \let\textlanguage\setlocale
876 \let\languagegettext\setlocale
```

## 4.2. Errors

### **\@nolanerr**

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

**\@noopterr** When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about `\PackageError` it must be  $\LaTeX 2_{\epsilon}$ , so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```
877 \edef\bbl@nulllanguage{\string\language=0}
878 \def\bbl@nocaption{\protect\bbl@nocaption@i}
879 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
880   \global\@namedef{#2}{\textbf{?#1?}}%
881   \@nameuse{#2}%
882   \edef\bbl@tempa{#1}%
883   \bbl@sreplace\bbl@tempa{name}{}}%
884   \bbl@warning{%
885     \@backslashchar#1 not set for '\language' name'. Please,\\%
886     define it after the language has been loaded\\%
887     (typically in the preamble) with:\\%
888     \string\setlocalecaption{\language name}{\bbl@tempa}{.}\\%
889     Feel free to contribute on github.com/latex3/babel.\\%
890     Reported}}
891 \def\bbl@tentative{\protect\bbl@tentative@i}
892 \def\bbl@tentative@i#1{%
893   \bbl@warning{%
894     Some functions for '#1' are tentative.\\%
895     They might not work as expected and their behavior\\%
896     could change in the future.\\%
897     Reported}}
898 \def\@nolanerr#1{\bbl@error{undefined-language}{#1}{}}
899 \def\@nopatterns#1{%
900   \bbl@warning
901     {No hyphenation patterns were preloaded for\\%
902     the language '#1' into the format.\\%
903     Please, configure your TeX system to add them and\\%
904     rebuild the format. Now I will use the patterns\\%
905     preloaded for \bbl@nulllanguage\space instead}}
906 \let\bbl@usehooks@gobbletwo
```



Here ended the now discarded switch.def.  
 Here also (currently) ends the base option.  
 907 \ifx\bbbl@onlyswitch\@empty\endinput\fi

### 4.3. More on selection

**\babelensure** The user command just parses the optional argument and creates a new macro named `\bbbl@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 `\bbbl@e@<language>` contains `\bbbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbbl@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.

```

908 \bbbl@trace{Defining babelensure}
909 \newcommand\babelensure[2][ ]{%
910   \AddBabelHook{babel-ensure}{afterextras}{%
911     \ifcase\bbbl@select@type
912       \bbbl@cl{e}%
913     \fi}%
914   \begingroup
915     \let\bbbl@ens@include\@empty
916     \let\bbbl@ens@exclude\@empty
917     \def\bbbl@ens@fontenc{\relax}%
918     \def\bbbl@tempb##1{%
919       \ifx\@empty##1\else\noexpand##1\expandafter\bbbl@tempb\fi}%
920     \edef\bbbl@tempa{\bbbl@tempb#1\@empty}%
921     \def\bbbl@tempb##1=##2\@{\@namedef{\bbbl@ens@##1}{##2}}%
922     \bbbl@foreach\bbbl@tempa{\bbbl@tempb##1\@}%
923     \def\bbbl@tempc{\bbbl@ensure}%
924     \expandafter\bbbl@add\expandafter\bbbl@tempc\expandafter{%
925       \expandafter{\bbbl@ens@include}}%
926     \expandafter\bbbl@add\expandafter\bbbl@tempc\expandafter{%
927       \expandafter{\bbbl@ens@exclude}}%
928     \toks@\expandafter{\bbbl@tempc}%
929     \bbbl@exp{%
930   \endgroup
931   \def<bbbl@e@#2>{\the\toks@{\bbbl@ens@fontenc}}%
932   \def\bbbl@ensure#1#2#3% 1: include 2: exclude 3: fontenc
933   \def\bbbl@tempb##1{% elt for (excluding) \bbbl@captionslist list
934     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
935       \edef##1{\noexpand\bbbl@nocaption
936         {\bbbl@stripslash##1}{\language\bbbl@stripslash##1}}%
937     \fi
938     \ifx##1\@empty\else
939       \in@{##1}{#2}%
940       \ifin@\else
941         \bbbl@ifunset{\bbbl@ensure@\language\name}%
942         {\bbbl@exp{%
943           \\DeclareRobustCommand\<bbbl@ensure@\language\name>[1]{%
944             \\foreignlanguage{\language\name}%
945             {\ifx\relax#3\else
946               \\fontencoding{#3}\\selectfont
947               \fi
948               #####1}}}%
949         }%
950         \toks@\expandafter{##1}%
951         \edef##1{%
952           \bbbl@csarg\noexpand{ensure@\language\name}%
953           {\the\toks@}}%
954       \fi

```

```

955     \expandafter\bbbl@tempb
956     \fi}%
957 \expandafter\bbbl@tempb\bbbl@captionslist\today\@empty
958 \def\bbbl@tempa##1{% elt for include list
959     \ifx##1\@empty\else
960         \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
961         \ifin\@else
962             \bbbl@tempb##1\@empty
963         \fi
964     \expandafter\bbbl@tempa
965     \fi}%
966 \bbbl@tempa##1\@empty}
967 \def\bbbl@captionslist{%
968 \prefacename\refname\abstractname\bibname\chaptername\appendixname
969 \contentsname\listfigurename\listtablename\indexname\figurename
970 \tablename\partname\enclname\ccname\headtoname\pagename\seename
971 \alsoname\proofname\glossaryname}

```

#### 4.4. 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.

```

972 \bbl@trace{Short tags}
973 \newcommand\babeltags[1]{%
974     \edef\bbbl@tempa{\zap@space#1 \@empty}%
975     \def\bbbl@tempb##1=##2\@{
976         \edef\bbbl@tempc{%
977             \noexpand\newcommand
978             \expandafter\noexpand\csname ##1\endcsname{%
979                 \noexpand\protect
980                 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
981             \noexpand\newcommand
982             \expandafter\noexpand\csname text##1\endcsname{%
983                 \noexpand\foreignlanguage{##2}}
984         \bbbl@tempc}%
985     \bbl@for\bbbl@tempa\bbbl@tempa{%
986         \expandafter\bbbl@tempb\bbbl@tempa\@{}}

```

#### 4.5. Compatibility with language.def

Plain e-TeX doesn't rely on language.dat, but babel can be made compatible with this format easily.

```

987 \bbl@trace{Compatibility with language.def}
988 \ifx\directlua\@undefined\else
989     \ifx\bbbl@luapatterns\@undefined
990         \input luabel.def
991     \fi
992 \fi
993 \ifx\bbbl@languages\@undefined
994     \ifx\directlua\@undefined
995         \openin1 = language.def % TODO. Remove hardcoded number
996         \ifeof1
997             \closein1
998             \message{I couldn't find the file language.def}
999         \else
1000             \closein1
1001             \begingroup
1002             \def\addlanguage#1#2#3#4#5{%
1003                 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1004                     \global\expandafter\let\csname l@#1\endcsname
1005                     \csname lang@#1\endcsname
1006                 \fi}%

```

```

1007     \def\uselanguage#1{%
1008     \input language.def
1009     \endgroup
1010     \fi
1011     \fi
1012     \chardef\l@english\z@
1013 \fi

```

**\addto** It takes two arguments, a *<control sequence>* and TeX-code to be added to the *<control sequence>*.

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

```

1014 \def\addto#1#2{%
1015   \ifx#1@undefined
1016     \def#1{#2}%
1017   \else
1018     \ifx#1\relax
1019       \def#1{#2}%
1020     \else
1021       {\toks@\expandafter{#1#2}%
1022        \xdef#1{\the\toks@}}%
1023     \fi
1024   \fi}

```

## 4.6. 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.

```

1025 \bbl@trace{Hooks}
1026 \newcommand\AddBabelHook[3][[%
1027   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{%
1028     \def\bbl@tempa##1,##3=#2,##3\@empty{\def\bbl@tempb{##2}}%
1029     \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1030     \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1031     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1032     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1033     \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1034 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1035 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1036 \def\bbl@usehooks{\bbl@usehooks@lang\languagename}
1037 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1038   \ifx\UseHook@undefined\else\UseHook{babel/*/#2}\fi
1039   \def\bbl@elth##1{%
1040     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1041     \bbl@cs{ev@#2@}%
1042     \ifx\languagename@undefined\else % Test required for Plain (?)
1043       \ifx\UseHook@undefined\else\UseHook{babel/#1/#2}\fi
1044       \def\bbl@elth##1{%
1045         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1#3}}%
1046         \bbl@cs{ev@#2@#1}%
1047       \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).

```

1048 \def\bbl@evargs{,% <- don't delete this comma
1049   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1050   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1051   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1052   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%

```

```

1053 beforestart=0, languagename=2, begindocument=1}
1054 \ifx\NewHook\@undefined\else % Test for Plain (?)
1055 \def\bbbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1056 \bbbl@foreach\bbbl@evargs{\bbbl@tempa#1\@@}
1057 \fi

```

Since the following command is meant for a hook (although a  $\LaTeX$  one), it's placed here.

```

1058 \providecommand\PassOptionsToLocale[2]{%
1059 \bbbl@csarg\bbbl@add@list{passto@#2}{#1}}

```

## 4.7. Setting up language files

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

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

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

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

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

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

Finally we check \originalTeX.

```

1060 \bbbl@trace{Macros for setting language files up}
1061 \def\bbbl@ldfinit{%
1062 \let\bbbl@screset\@empty
1063 \let\BabelStrings\bbbl@opt@string
1064 \let\BabelOptions\@empty
1065 \let\BabelLanguages\relax
1066 \ifx\originalTeX\@undefined
1067 \let\originalTeX\@empty
1068 \else
1069 \originalTeX
1070 \fi}
1071 \def\LdfInit#1#2{%
1072 \chardef\atcatcode=\catcode`\@
1073 \catcode`\@=11\relax
1074 \chardef\eqcatcode=\catcode`\=
1075 \catcode`\==12\relax
1076 \expandafter\if\expandafter\@backslashchar
1077 \expandafter\@car\string#2\@nil
1078 \ifx#2\@undefined\else
1079 \ldf@quit{#1}%
1080 \fi
1081 \else
1082 \expandafter\ifx\csname#2\endcsname\relax\else
1083 \ldf@quit{#1}%
1084 \fi
1085 \fi
1086 \bbbl@ldfinit}

```

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

```

1087 \def\ldf@quit#1{%
1088 \expandafter\main@language\expandafter{#1}%
1089 \catcode`\@=\atcatcode \let\atcatcode\relax

```

```

1090 \catcode`\==\eqcatcode \let\eqcatcode\relax
1091 \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.

```

1092 \def\bbl@afterldf#1{%^^A TODO. #1 is not used. Remove
1093 \bbl@afterlang
1094 \let\bbl@afterlang\relax
1095 \let\BabelModifiers\relax
1096 \let\bbl@screset\relax}%
1097 \def\ldf@finish#1{%
1098 \loadlocalcfg{#1}%
1099 \bbl@afterldf{#1}%
1100 \expandafter\main@language\expandafter{#1}%
1101 \catcode`\@=\atcatcode \let\atcatcode\relax
1102 \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  $\LaTeX$ .

```

1103 \@onlypreamble\LdfInit
1104 \@onlypreamble\ldf@quit
1105 \@onlypreamble\ldf@finish

```

### **\main@language**

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

```

1106 \def\main@language#1{%
1107 \def\bbl@main@language{#1}%
1108 \let\languagename\bbl@main@language
1109 \let\localename\bbl@main@language
1110 \let\mainlocalename\bbl@main@language
1111 \bbl@id@assign
1112 \bbl@patterns{\languagename}}

```

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

The code written to the aux file attempts to avoid errors if babel is removed from the document.

```

1113 \def\bbl@beforestart{%
1114 \def\@nolanerr##1{%
1115 \bbl@carg\chardef{l@##1}\z@
1116 \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1117 \bbl@usehooks{beforestart}{}%
1118 \global\let\bbl@beforestart\relax}
1119 \AtBeginDocument{%
1120 {\@nameuse{bbl@beforestart}}% Group!
1121 \if@filesw
1122 \providecommand\babel@aux[2]{}%
1123 \immediate\write\@mainaux{\unexpanded{%
1124 \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1125 \immediate\write\@mainaux{string\@nameuse{bbl@beforestart}}}%
1126 \fi
1127 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1128 \ifbbl@single % must go after the line above.
1129 \renewcommand\selectlanguage[1]{}%
1130 \renewcommand\foreignlanguage[2]{#2}%
1131 \global\let\babel@aux\@gobbletwo % Also as flag
1132 \fi}

```

```

1133 %
1134 \ifcase\bbl@engine\or
1135 \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1136 \fi

```

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

```

1137 \def\select@language@x#1{%
1138 \ifcase\bbl@select@type
1139 \bbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1140 \else
1141 \select@language{#1}%
1142 \fi}

```

## 4.8. Shorthands

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.

```

1143 \bbl@trace{Shorhands}
1144 \def\bbl@withactive#1#2{%
1145 \begingroup
1146 \lccode`~=`#2\relax
1147 \lowercase{\endgroup#1~}}

```

**`\bbl@add@special`** The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if  $\TeX$  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.

```

1148 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1149 \bbl@add@dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1150 \bbl@ifunset{\@sanitize}{\bbl@add@\@sanitize{\makeother#1}}%
1151 \ifx\nfss@catcodes@\undefined\else % TODO - same for above
1152 \begingroup
1153 \catcode`#1\active
1154 \nfss@catcodes
1155 \ifnum\catcode`#1=\active
1156 \endgroup
1157 \bbl@add\nfss@catcodes{\makeother#1}%
1158 \else
1159 \endgroup
1160 \fi
1161 \fi}

```

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

For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines " as `\active@prefix "\active@char` (where the first " is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (i.e., with the original "); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (e.g., `\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).

```

1162 \def\bbl@active@def#1#2#3#4{%
1163   \@namedef{#3#1}{%
1164     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1165     \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1166     \else
1167     \bbl@afterfi\csname#2@sh@#1@\endcsname
1168     \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.

```

1169   \long\@namedef{#3@arg#1}##1{%
1170     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1171     \bbl@afterelse\csname#4#1\endcsname##1%
1172     \else
1173     \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1174     \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.

```

1175 \def\initiate@active@char#1{%
1176   \bbl@ifunset{active@char\string#1}%
1177   {\bbl@withactive
1178    {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1179   {}}

```

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).

```

1180 \def\@initiate@active@char#1#2#3{%
1181   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1182   \ifx#1\@undefined
1183     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1184   \else
1185     \bbl@csarg\let{oridef@#2}#1%
1186     \bbl@csarg\edef{oridef@#2}{%
1187       \let\noexpand#1%
1188       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1189   \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`).

```

1190   \ifx#1#3\relax
1191     \expandafter\let\csname normal@char#2\endcsname#3%
1192   \else
1193     \bbl@info{Making #2 an active character}%
1194     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1195     \@namedef{normal@char#2}{%
1196       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1197   \else
1198     \@namedef{normal@char#2}{#3}%
1199   \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).

```

1200   \bbl@restoreactive{#2}%
1201   \AtBeginDocument{%

```

```

1202     \catcode`#2\active
1203     \if@filesw
1204         \immediate\write\@mainaux{\catcode`\string#2\active}%
1205     \fi}%
1206     \expandafter\bbbl@add@special\csname#2\endcsname
1207     \catcode`#2\active
1208 \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⟩`).

```

1209 \let\bbbl@tempa\@firstoftwo
1210 \if\string^#2%
1211     \def\bbbl@tempa{\noexpand\textormath}%
1212 \else
1213     \ifx\bbbl@mathnormal\@undefined\else
1214         \let\bbbl@tempa\bbbl@mathnormal
1215     \fi
1216 \fi
1217 \expandafter\edef\csname active@char#2\endcsname{%
1218     \bbbl@tempa
1219         {\noexpand\if@safe@actives
1220             \noexpand\expandafter
1221             \expandafter\noexpand\csname normal@char#2\endcsname
1222             \noexpand\else
1223             \noexpand\expandafter
1224             \expandafter\noexpand\csname bbl@doactive#2\endcsname
1225             \noexpand\fi}%
1226     {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1227 \bbbl@csarg\edef{doactive#2}{%
1228     \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

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

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

```

1229 \bbbl@csarg\edef{active@#2}{%
1230     \noexpand\active@prefix\noexpand#1%
1231     \expandafter\noexpand\csname active@char#2\endcsname}%
1232 \bbbl@csarg\edef{normal@#2}{%
1233     \noexpand\active@prefix\noexpand#1%
1234     \expandafter\noexpand\csname normal@char#2\endcsname}%
1235 \bbbl@ncarg\let#1\bbbl@normal@#2}%

```

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.

```

1236 \bbbl@active@def#2\user@group{user@active}{language@active}%
1237 \bbbl@active@def#2\language@group{language@active}{system@active}%
1238 \bbbl@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.

```

1239 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1240     {\expandafter\noexpand\csname normal@char#2\endcsname}%
1241 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1242     {\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 `\prim@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.

```
1243 \if\string'#2%
1244   \let\prim@s\bb@prim@s
1245   \let\active@math@prime#1%
1246 \fi
1247 \bb@usehooks{initiateactive}{{#1}{#2}{#3}}
```

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

```
1248 <<{*More package options}>> ≡
1249 \DeclareOption{math=active}{}
1250 \DeclareOption{math=normal}{\def\bb@mathnormal{\noexpand\textormath}}
1251 <</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.

```
1252 \ifpackagewith{babel}{KeepShorthandsActive}%
1253   {\let\bb@restoreactive\@gobble}%
1254   {\def\bb@restoreactive#1{%
1255     \bb@exp{%
1256       \\AfterBabelLanguage\\CurrentOption
1257       {\catcode`#1=\the\catcode`#1\relax}%
1258       \\AtEndOfPackage
1259       {\catcode`#1=\the\catcode`#1\relax}}}%
1260   \AtEndOfPackage{\let\bb@restoreactive\@gobble}}
```

**\bb@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 `\bb@firstcs` or `\bb@scndcs`. Hence two more arguments need to follow it.

```
1261 \def\bb@sh@select#1#2{%
1262   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1263     \bb@afterelse\bb@scndcs
1264   \else
1265     \bb@afterfi\csname#1@sh@#2@sel\endcsname
1266   \fi}
```

**\active@prefix** 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.

```
1267 \begingroup
1268 \bb@ifunset{ifincsname}%^^A Ugly. Correct? Only Plain?
1269 {\gdef\active@prefix#1{%
1270   \ifx\protect\@typeset@protect
1271   \else
1272     \ifx\protect\@unexpandable@protect
1273       \noexpand#1%
1274     \else
1275       \protect#1%
1276     \fi
1277   \expandafter\@gobble
1278   \fi}}
1279 {\gdef\active@prefix#1{%
1280   \ifincsname
```

```

1281     \string#1%
1282     \expandafter\@gobble
1283     \else
1284     \ifx\protect\@typeset@protect
1285     \else
1286     \ifx\protect\@unexpandable@protect
1287     \noexpand#1%
1288     \else
1289     \protect#1%
1290     \fi
1291     \expandafter\expandafter\expandafter\@gobble
1292     \fi
1293     \fi}}
1294 \endgroup

```

**if@safe@actives** In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ‘other’) 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⟩. When this expansion mode is active (with \@safe@activetrue), something like "13"13 becomes "12"12 in an \edef (in other words, shorthands are \string’ed). This contrasts with \protected@edef, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with \@safe@activefalse).

```

1295 \newif\if@safe@actives
1296 \@safe@activefalse

```

**\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.

```

1297 \def\bbl@restore@actives{\if@safe@actives\@safe@activefalse\fi}

```

#### **\bbl@activate**

**\bbl@deactivate** 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.

```

1298 \chardef\bbl@activated\z@
1299 \def\bbl@activate#1{%
1300   \chardef\bbl@activated\@ne
1301   \bbl@withactive{\expandafter\let\expandafter}#1%
1302   \csname bbl@active@\string#1\endcsname}
1303 \def\bbl@deactivate#1{%
1304   \chardef\bbl@activated\tw@
1305   \bbl@withactive{\expandafter\let\expandafter}#1%
1306   \csname bbl@normal@\string#1\endcsname}

```

#### **\bbl@firstcs**

**\bbl@scndcs** These macros are used only as a trick when declaring shorthands.

```

1307 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1308 \def\bbl@scndcs#1#2{\csname#2\endcsname}

```

**\declare@shorthand** 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 T<sub>E</sub>X code in text mode, (2) the string for hyperref, (3) the T<sub>E</sub>X 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.

```

1309 \def\babel@texpdf#1#2#3#4{%

```

```

1310 \ifx\texorpdfstring\undefined
1311   \textormath{#1}{#3}%
1312 \else
1313   \texorpdfstring{\textormath{#1}{#3}}{#2}%
1314   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1315 \fi}
1316 %
1317 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1318 \def\@decl@short#1#2#3\@nil#4{%
1319   \def\bbl@tempa{#3}%
1320   \ifx\bbl@tempa\@empty
1321     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1322     \bbl@ifunset{#1@sh@\string#2@}{}%
1323     {\def\bbl@tempa{#4}%
1324       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1325       \else
1326         \bbl@info
1327           {Redefining #1 shorthand \string#2\}%
1328           in language \CurrentOption}%
1329     \fi}%
1330   \@namedef{#1@sh@\string#2@}{#4}%
1331 \else
1332   \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1333   \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1334   {\def\bbl@tempa{#4}%
1335     \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1336     \else
1337       \bbl@info
1338         {Redefining #1 shorthand \string#2\string#3\}%
1339         in language \CurrentOption}%
1340     \fi}%
1341   \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1342 \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.

```

1343 \def\textormath{%
1344   \ifmmode
1345     \expandafter\@secondoftwo
1346   \else
1347     \expandafter\@firstoftwo
1348   \fi}

```

### **\user@group**

### **\language@group**

**\system@group** The current concept of ‘shorthands’ supports three levels or groups of shorthands.

For each level the name of the level or group is stored in a macro. The default is to have a user group; use language group ‘english’ and have a system group called ‘system’.

```

1349 \def\user@group{user}
1350 \def\language@group{english} %^^A I don't like defaults
1351 \def\system@group{system}

```

**\useshorthands** This is the user level macro. It initializes and activates the character for use as a shorthand character (i.e., 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.

```

1352 \def\useshorthands{%
1353   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1354 \def\bbl@usesh@s#1{%
1355   \bbl@usesh@x
1356   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1357   {#1}}

```

```

1358 \def\bbl@usesh@x#1#2{%
1359   \bbl@ifshorthand{#2}%
1360   {\def\user@group{user}%
1361     \initiate@active@char{#2}%
1362     #1%
1363     \bbl@activate{#2}}%
1364   {\bbl@error{shorthand-is-off}{#2}{}}}

```

**\defineshorthand** Currently we only support two groups of user level shorthands, named internally `user` and `user@(language)` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1365 \def\user@language@group{user@\language@group}
1366 \def\bbl@set@user@generic#1#2{%
1367   \bbl@ifunset{user@generic@active#1}%
1368   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1369     \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1370     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1371       \expandafter\noexpand\csname normal@char#1\endcsname}%
1372     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1373       \expandafter\noexpand\csname user@active#1\endcsname}}%
1374   \@empty}
1375 \newcommand\defineshorthand[3][user]{%
1376   \edef\bbl@tempa{\zap@space#1 \@empty}%
1377   \bbl@for\bbl@tempb\bbl@tempa{%
1378     \if*\expandafter\@car\bbl@tempb\@nil
1379     \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1380     \@expandtwoargs
1381     \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1382   \fi
1383   \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

**\languageshorthands** A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed.

```

1384 \def\languageshorthands#1{%
1385   \bbl@ifsamestring{none}{#1}{}%
1386   \bbl@once{short-\localename-#1}{%
1387     \bbl@info{'\localename' activates '#1' shorthands.\Reported }}}}
1388 \def\language@group{#1}

```

**\aliasshorthand** *Deprecated*. 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 latter to `\active@char`".

```

1389 \def\aliasshorthand#1#2{%
1390   \bbl@ifshorthand{#2}%
1391   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1392     \if\document\@notprerr
1393       \@notshorthand{#2}%
1394     \else
1395       \initiate@active@char{#2}%
1396       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1397       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1398       \bbl@activate{#2}%
1399     \fi
1400   \fi}%
1401   {\bbl@error{shorthand-is-off}{#2}{}}}

```

**\@notshorthand**

```

1402 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}

```

## **\shorthandon**

**\shorthandoff** The first level definition of these macros just passes the argument on to `\bbl@switch@sh`, adding `\@nil` at the end to denote the end of the list of characters.

```
1403 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1404 \DeclareRobustCommand*\shorthandoff{%
1405   \ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1406 \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.

```
1407 \def\bbl@switch@sh#1#2{%
1408   \ifx#2\@nnil\else
1409     \bbl@ifunset{bbl@active@\string#2}%
1410     {\bbl@error{not-a-shorthand-b}{\#2}}}%
1411     {\ifcase#1%   off, on, off*
1412       \catcode`#2\relax
1413       \or
1414       \catcode`#2\active
1415       \bbl@ifunset{bbl@shdef@\string#2}%
1416       {}%
1417       {\bbl@withactive{\expandafter\let\expandafter}#2%
1418         \csname bbl@shdef@\string#2\endcsname
1419         \bbl@csarg\let{shdef@\string#2}\relax}%
1420       \ifcase\bbl@activated\or
1421         \bbl@activate{#2}%
1422       \else
1423         \bbl@deactivate{#2}%
1424       \fi
1425       \or
1426       \bbl@ifunset{bbl@shdef@\string#2}%
1427       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1428       {}%
1429       \csname bbl@oricat@\string#2\endcsname
1430       \csname bbl@oridef@\string#2\endcsname
1431       \fi}%
1432   \bbl@afterfi\bbl@switch@sh#1%
1433 \fi}
```

Note the value is that at the expansion time; e.g., in the preamble shorthands are usually deactivated.

```
1434 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1435 \def\bbl@putsh#1{%
1436   \bbl@ifunset{bbl@active@\string#1}%
1437   {\bbl@putsh@i#1\@empty\@nnil}%
1438   {\csname bbl@active@\string#1\endcsname}}
1439 \def\bbl@putsh@i#1#2\@nnil{%
1440   \csname\language@group @sh@\string#1@%
1441     \ifx\@empty#2\else\string#2@\fi\endcsname}
1442 %
1443 \ifx\bbl@opt@shorthands\@nnil\else
1444   \let\bbl@s@initiate@active@char\initiate@active@char
1445   \def\initiate@active@char#1{%
1446     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1447   \let\bbl@s@switch@sh\bbl@switch@sh
1448   \def\bbl@switch@sh#1#2{%
1449     \ifx#2\@nnil\else
```

```

1450     \bbl@afterfi
1451     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1452     \fi}
1453     \let\bbl@s@activate\bbl@activate
1454     \def\bbl@activate#1{%
1455       \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1456     \let\bbl@s@deactivate\bbl@deactivate
1457     \def\bbl@deactivate#1{%
1458       \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1459     \fi

```

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

```

1460 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@string#1}{#3}{#2}}

```

### **\bbl@prim@s**

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

```

1461 \def\bbl@prim@s{%
1462   \prime\futurelet\@let@token\bbl@pr@m@s}
1463 \def\bbl@if@primes#1#2{%
1464   \ifx#1\@let@token
1465     \expandafter\@firstoftwo
1466   \else\ifx#2\@let@token
1467     \bbl@afterelse\expandafter\@firstoftwo
1468   \else
1469     \bbl@afterfi\expandafter\@secondoftwo
1470   \fi\fi}
1471 \begingroup
1472 \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1473 \catcode`\'=12 \catcode`\"=\active \lccode`\"='\`
1474 \lowercase{%
1475   \gdef\bbl@pr@m@s{%
1476     \bbl@if@primes" '%
1477     \pr@@@s
1478     {\bbl@if@primes*\^pr@@@t\egroup}}
1479 \endgroup

```

Usually the `~` is active and expands to `\penalty\M\L`. 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).

```

1480 \initiate@active@char{~}
1481 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1482 \bbl@activate{~}

```

### **\OT1dqpos**

**\T1dqpos** 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.

```

1483 \expandafter\def\csname OT1dqpos\endcsname{127}
1484 \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

```

1485 \ifx\f@encoding\undefined
1486   \def\f@encoding{OT1}
1487 \fi

```

## 4.9. 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.

```
1488 \bbl@trace{Language attributes}
1489 \newcommand\languageattribute[2]{%
1490   \def\bbl@tempc{#1}%
1491   \bbl@fixname\bbl@tempc
1492   \bbl@iflanguage\bbl@tempc{%
1493     \bbl@vforeach{#2}{%
```

To make sure each attribute is selected only once, we store the already selected attributes in `\bbl@known@attrs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```
1494     \ifx\bbl@known@attrs\undefined
1495       \in@false
1496     \else
1497       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1498     \fi
1499     \ifin@
1500       \bbl@warning{%
1501         You have more than once selected the attribute '##1'\%
1502         for language #1. Reported}%
1503     \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.

```
1504       \bbl@exp{%
1505         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1506       \edef\bbl@tempa{\bbl@tempc-##1}%
1507       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1508       {\csname\bbl@tempc @attr##1\endcsname}%
1509       {\@attrerr{\bbl@tempc}{##1}}%
1510     \fi}}
1511 \@onlypreamble\languageattribute
```

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

```
1512 \newcommand*\@attrerr[2]{%
1513   \bbl@error{unknown-attribute}{#1}{#2}{}}
```

**\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}`.

```
1514 \def\bbl@declare@ttribute#1#2#3{%
1515   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1516   \ifin@
1517     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1518   \fi
1519   \bbl@add@list\bbl@attributes{#1-#2}%
1520   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

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

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

```

1521 \def\bbl@ifattributeset#1#2#3#4{%
1522   \ifx\bbl@known@attrs\@undefined
1523     \in@false
1524   \else
1525     \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1526   \fi
1527   \ifin@
1528     \bbl@afterelse#3%
1529   \else
1530     \bbl@afterfi#4%
1531   \fi}

```

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

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```

1532 \def\bbl@ifknown@ttrib#1#2{%
1533   \let\bbl@tempa\@secondoftwo
1534   \bbl@loopx\bbl@tempb{#2}{%
1535     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1536     \ifin@
1537       \let\bbl@tempa\@firstoftwo
1538     \else
1539       \fi}%
1540   \bbl@tempa}

```

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

```

1541 \def\bbl@clear@ttribs{%
1542   \ifx\bbl@attributes\@undefined\else
1543     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1544       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1545     \let\bbl@attributes\@undefined
1546   \fi}
1547 \def\bbl@clear@ttrib#1-#2.{%
1548   \expandafter\let\csname#1@attr#2\endcsname\@undefined}
1549 \AtBeginDocument{\bbl@clear@ttribs}

```

## 4.10. Support for saving and redefining macros

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**

**\babel@beginsave** The initialization of a new save cycle: reset the counter to zero.

```

1550 \bbl@trace{Macros for saving definitions}
1551 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1552 \newcount\babel@savecnt
1553 \babel@beginsave

```

**\babel@save**



**\babel@savevariable** The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX` (which has to be expandable, i.e., you shouldn't let it to `\relax`). 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. To avoid messing saved definitions up, they are saved only the very first time.

```

1554 \def\babel@save#1{%
1555   \def\bbl@tempa{⟨#1,⟩}% Clumsy, for Plain
1556   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1557     \expandafter{\expandafter,\bbl@savextras,}%
1558     \expandafter\in@\bbl@tempa
1559   \ifin@%else
1560     \bbl@add\bbl@savextras{⟨#1,⟩}%
1561     \bbl@carg\let{\babel@number\babel@savecnt}#1\relax
1562     \toks@\expandafter{\originalTeX\let#1=}%
1563     \bbl@exp{%
1564       \def\\originalTeX{\the\toks@⟨\babel@number\babel@savecnt⟩\relax}}%
1565     \advance\babel@savecnt@ne
1566   \fi}
1567 \def\babel@savevariable#1{%
1568   \toks@\expandafter{\originalTeX #1}%
1569   \bbl@exp{\def\\originalTeX{\the\toks@⟨the#1\relax⟩}}

```

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

```

1570 \def\bbl@redefine#1{%
1571   \edef\bbl@tempa{\bbl@stripslash#1}%
1572   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1573   \expandafter\def\csname\bbl@tempa\endcsname}
1574 \@onlypreamble\bbl@redefine

```

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

```

1575 \def\bbl@redefine@long#1{%
1576   \edef\bbl@tempa{\bbl@stripslash#1}%
1577   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1578   \long\expandafter\def\csname\bbl@tempa\endcsname}
1579 \@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_`.

```

1580 \def\bbl@redefineroobust#1{%
1581   \edef\bbl@tempa{\bbl@stripslash#1}%
1582   \bbl@ifunset{\bbl@tempa\space}%
1583     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1584       \bbl@exp{\def\\#1\\protect\<\bbl@tempa\space>}}}%
1585     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1586     \@namedef{\bbl@tempa\space}
1587 \@onlypreamble\bbl@redefineroobust

```

## 4.11. French spacing

**\bbl@frenchspacing**

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

```

1588 \def\bbl@frenchspacing{%
1589   \ifnum\the\sfcode`\.=\@m
1590     \let\bbl@nonfrenchspacing\relax
1591   \else
1592     \frenchspacing
1593     \let\bbl@nonfrenchspacing\nonfrenchspacing
1594   \fi}
1595 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

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.

```

1596 \let\bbl@elt\relax
1597 \edef\bbl@fs@chars{%
1598   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1599   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1600   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1601 \def\bbl@pre@fs{%
1602   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1603   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1604 \def\bbl@post@fs{%
1605   \bbl@save@sfcodes
1606   \edef\bbl@tempa{\bbl@cl{frspc}}%
1607   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1608   \if u\bbl@tempa      % do nothing
1609   \else\if n\bbl@tempa % non french
1610     \def\bbl@elt##1##2##3{%
1611       \ifnum\sfcode`##1=##2\relax
1612         \babel@savevariable{\sfcode`##1}%
1613         \sfcode`##1=##3\relax
1614       \fi}%
1615     \bbl@fs@chars
1616   \else\if y\bbl@tempa % french
1617     \def\bbl@elt##1##2##3{%
1618       \ifnum\sfcode`##1=##3\relax
1619         \babel@savevariable{\sfcode`##1}%
1620         \sfcode`##1=##2\relax
1621       \fi}%
1622     \bbl@fs@chars
1623   \fi\fi\fi}

```

## 4.12. Hyphens

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

```

1624 \bbl@trace{Hyphens}
1625 \@onlypreamble\babelhyphenation
1626 \AtEndOfPackage{%
1627   \newcommand\babelhyphenation[2][\@empty]{%
1628     \ifx\bbl@hyphenation@\relax
1629       \let\bbl@hyphenation@\@empty
1630     \fi
1631     \ifx\bbl@hyphlist\@empty\else
1632       \bbl@warning{%
1633         You must not intermingle \string\selectlanguage\space and\%
1634         \string\babelhyphenation\space or some exceptions will not\%
1635         be taken into account. Reported}%
1636     \fi

```

```

1637 \ifx\@empty#1%
1638 \protected@edef\bb@hyphenation@{\bb@hyphenation@space#2}%
1639 \else
1640 \bb@vforeach{#1}{%
1641 \def\bb@tempa{##1}%
1642 \bb@fixname\bb@tempa
1643 \bb@iflanguage\bb@tempa{%
1644 \bb@csarg\protected@edef{hyphenation@\bb@tempa}{%
1645 \bb@ifunset{bb@hyphenation@\bb@tempa}%
1646 }%
1647 {\csname bb@hyphenation@\bb@tempa\endcsname space}%
1648 #2}}%
1649 \fi}}

```

**\babelhyphenmins** Only  $\LaTeX$  (basically because it's defined with a  $\LaTeX$  tool).

```

1650 \ifx\NewDocumentCommand\@undefined\else
1651 \NewDocumentCommand\babelhyphenmins{sommo}{%
1652 \IfNoValueTF{#2}%
1653 {\protected@edef\bb@hyphenmins@{\set@hyphenmins{#3}{#4}}%
1654 \IfValueT{#5}{%
1655 \protected@edef\bb@hyphenatmin@{\hyphenationmin=#5\relax}}%
1656 \IfBooleanT{#1}{%
1657 \left@hyphenmin=#3\relax
1658 \right@hyphenmin=#4\relax
1659 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1660 {\edef\bb@tempb{\zap@space#2 \@empty}%
1661 \bb@for\bb@tempa\bb@tempb{%
1662 \@namedef{bb@hyphenmins@\bb@tempa}{\set@hyphenmins{#3}{#4}}%
1663 \IfValueT{#5}{%
1664 \@namedef{bb@hyphenatmin@\bb@tempa}{\hyphenationmin=#5\relax}}}%
1665 \IfBooleanT{#1}{\bb@error{hyphenmins-args}{}}}}
1666 \fi

```

**\bb@allowhyphens** This macro makes hyphenation possible. Basically its definition is nothing more than `\nbreak \hskip 0pt plus 0pt`.  $\TeX$  begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1667 \def\bb@allowhyphens{\ifvmode\else\nbreak\hskip\z@skip\fi}
1668 \def\bb@t@one{T1}
1669 \def\allowhyphens{\ifx\cf@encoding\bb@t@one\else\bb@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`.

```

1670 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1671 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1672 \def\bb@hyphen{%
1673 \ifstar{\bb@hyphen@i @}{\bb@hyphen@i\@empty}}
1674 \def\bb@hyphen@i#1#2{%
1675 \lowercase{\bb@ifunset{bb@hy@#1#2\@empty}}%
1676 {\csname bb@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1677 {\lowercase{\csname bb@hy@#1#2\@empty\endcsname}}}

```

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

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

```

1678 \def\bb@usehyphen#1{%
1679 \leavevmode

```

```

1680 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1681 \nobreak\hskip\z@skip}
1682 \def\bbl@usehyphen#1{%
1683 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1684 \def\bbl@hyphenchar{%
1685 \ifnum\hyphenchar\font=\m@ne
1686 \babelnullhyphen
1687 \else
1688 \char\hyphenchar\font
1689 \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.

```

1690 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1691 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1692 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1693 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1694 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1695 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1696 \def\bbl@hy@repeat{%
1697 \bbl@usehyphen{%
1698 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1699 \def\bbl@hy@repeat{%
1700 \bbl@usehyphen{%
1701 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1702 \def\bbl@hy@empty{\hskip\z@skip}
1703 \def\bbl@hy@empty{\discretionary{}{}}{}}

```

**\bbl@disc** For some languages the macro `\bbl@disc` is used to ease the insertion of discretionary for letters that behave ‘abnormally’ at a breakpoint.

```

1704 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

### 4.13. 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 tool. It makes global a local variable. This is not the best solution, but it works.

```

1705 \bbl@trace{Multiencoding strings}
1706 \def\bbl@tglobal#1{\global\let#1#1}

```

The following option is currently no-op. It was meant for the deprecated `\SetCase`.

```

1707 <<{*More package options}>> ≡
1708 \DeclareOption{nocase}{}
1709 <</More package options>>

```

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

```

1710 <<{*More package options}>> ≡
1711 \let\bbl@opt@strings\@nnil % accept strings=value
1712 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1713 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1714 \def\BabelStringsDefault{generic}
1715 <</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.

```

1716 \@onlypreamble\StartBabelCommands
1717 \def\StartBabelCommands{%
1718   \begingroup
1719   \@tempcnta="7F
1720   \def\bbbl@tempa{%
1721     \ifnum\@tempcnta>"FF\else
1722       \catcode\@tempcnta=11
1723       \advance\@tempcnta\@ne
1724       \expandafter\bbbl@tempa
1725     \fi}%
1726   \bbbl@tempa
1727   <@Macros local to BabelCommands@>
1728   \def\bbbl@provstring##1##2{%
1729     \providecommand###1{##2}%
1730     \bbbl@tglobal##1}%
1731   \global\let\bbbl@scafter\@empty
1732   \let\StartBabelCommands\bbbl@startcmds
1733   \ifx\BabelLanguages\relax
1734     \let\BabelLanguages\CurrentOption
1735   \fi
1736   \begingroup
1737   \let\bbbl@screset\@nnil % local flag - disable 1st stopcommands
1738   \StartBabelCommands}
1739 \def\bbbl@startcmds{%
1740   \ifx\bbbl@screset\@nnil\else
1741     \bbbl@usehooks{stopcommands}{}%
1742   \fi
1743   \endgroup
1744   \begingroup
1745   \@ifstar
1746     {\ifx\bbbl@opt@strings\@nnil
1747       \let\bbbl@opt@strings\BabelStringsDefault
1748     \fi
1749     \bbbl@startcmds@i}%
1750   \bbbl@startcmds@i}
1751 \def\bbbl@startcmds@i##1##2{%
1752   \edef\bbbl@L{\zap@space#1 \@empty}%
1753   \edef\bbbl@G{\zap@space#2 \@empty}%
1754   \bbbl@startcmds@ii}
1755 \let\bbbl@startcommands\StartBabelCommands

```

Parse the encoding info to get the label, input, and font parts.

Select the behavior of `\SetString`. There are two main cases, depending of if there is an optional argument: without it and `strings=encoded`, strings are defined always; otherwise, they are set only if they are still undefined (i.e., 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 (i.e., 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.

```

1756 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1757   \let\SetString\@gobbletwo
1758   \let\bbbl@stringdef\@gobbletwo
1759   \let\AfterBabelCommands\@gobble
1760   \ifx\@empty#1%
1761     \def\bbbl@sc@label{generic}%
1762     \def\bbbl@encstring##1##2{%
1763       \ProvideTextCommandDefault###1{##2}%
1764       \bbbl@tglobal##1%
1765       \expandafter\bbbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1766 \let\bbL@sctest\in@true
1767 \else
1768 \let\bbL@sc@charset\space % <- zapped below
1769 \let\bbL@sc@fontenc\space % <- " "
1770 \def\bbL@tempa##1=##2\@nil{%
1771 \bbL@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1772 \bbL@vforeach{label=#1}{\bbL@tempa##1\@nil}%
1773 \def\bbL@tempa##1 ##2{% space -> comma
1774 ##1%
1775 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbL@afterfi\bbL@tempa##2\fi}%
1776 \edef\bbL@sc@fontenc{\expandafter\bbL@tempa\bbL@sc@fontenc\@empty}%
1777 \edef\bbL@sc@label{\expandafter\zap@space\bbL@sc@label\@empty}%
1778 \edef\bbL@sc@charset{\expandafter\zap@space\bbL@sc@charset\@empty}%
1779 \def\bbL@encstring##1##2{%
1780 \bbL@foreach\bbL@sc@fontenc{%
1781 \bbL@ifunset{T@####1}%
1782 }}%
1783 {\ProvideTextCommand##1{####1}{##2}%
1784 \bbL@tglobal##1%
1785 \expandafter
1786 \bbL@tglobal\csname####1\string##1\endcsname}}}%
1787 \def\bbL@sctest{%
1788 \bbL@xin@{\bbL@opt@strings,}{,\bbL@sc@label,\bbL@sc@fontenc,}}%
1789 \fi
1790 \ifx\bbL@opt@strings\@nnil % i.e., no strings key -> defaults
1791 \else\ifx\bbL@opt@strings\relax % i.e., strings=encoded
1792 \let\AfterBabelCommands\bbL@aftercmds
1793 \let\SetString\bbL@setstring
1794 \let\bbL@stringdef\bbL@encstring
1795 \else % i.e., strings=value
1796 \bbL@sctest
1797 \ifin@
1798 \let\AfterBabelCommands\bbL@aftercmds
1799 \let\SetString\bbL@setstring
1800 \let\bbL@stringdef\bbL@provstring
1801 \fi\fi\fi
1802 \bbL@scswitch
1803 \ifx\bbL@G\@empty
1804 \def\SetString##1##2{%
1805 \bbL@error{missing-group}{##1}{}}}%
1806 \fi
1807 \ifx\@empty#1%
1808 \bbL@usehooks{defaultcommands}{}%
1809 \else
1810 \@expandtwoargs
1811 \bbL@usehooks{encodedcommands}{\bbL@sc@charset}{\bbL@sc@fontenc}}%
1812 \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).

```

1813 \def\bbL@forlang#1#2{%
1814 \bbL@for#1\bbL@L{%
1815 \bbL@xin@{,#1,}{,\BabelLanguages,}%
1816 \ifin@#2\relax\fi}}
1817 \def\bbL@scswitch{%
1818 \bbL@forlang\bbL@tempa{%
1819 \ifx\bbL@G\@empty\else

```

```

1820     \ifx\SetString@gobbletwo\else
1821     \edef\bbL@GL{\bbL@G\bbL@tempa}%
1822     \bbL@xin@{\, \bbL@GL,}{, \bbL@screset,}%
1823     \ifin@else
1824     \global\expandafter\let\csname\bbL@GL\endcsname@undefined
1825     \xdef\bbL@screset{\bbL@screset, \bbL@GL}%
1826     \fi
1827     \fi
1828     \fi}}
1829 \AtEndOfPackage{%
1830   \def\bbL@forlang#1#2{\bbL@for#1\bbL@L{\bbL@ifunset{date#1}{}{#2}}}%
1831   \let\bbL@scswitch\relax}
1832 \onlypreamble\EndBabelCommands
1833 \def\EndBabelCommands{%
1834   \bbL@usehooks{stopcommands}{}%
1835   \endgroup
1836   \endgroup
1837   \bbL@scafter}
1838 \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 (i.e., 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.

```

1839 \def\bbL@setstring#1#2{% e.g., \prefacename{<string>}
1840   \bbL@forlang\bbL@tempa{%
1841     \edef\bbL@LC{\bbL@tempa\bbL@stripslash#1}%
1842     \bbL@ifunset{\bbL@LC}% e.g., \germanchaptername
1843     {\bbL@exp{%
1844       \global\bbL@add\<\bbL@G\bbL@tempa>{\bbL@scset\#1\<\bbL@LC>}}}%
1845     }%
1846     \def\BabelString{#2}%
1847     \bbL@usehooks{stringprocess}{}%
1848     \expandafter\bbL@stringdef
1849     \csname\bbL@LC\expandafter\endcsname\expandafter{\BabelString}}

```

A little auxiliary command sets the string. Formerly used with casing. Very likely no longer necessary, although it’s used in `\setlocalecaption`.

```

1850 \def\bbL@scset#1#2{\def#1{#2}}

```

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.

```

1851 <<{*Macros local to BabelCommands}>> ≡
1852 \def\SetStringLoop##1##2{%
1853   \def\bbL@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1854   \count@\z@
1855   \bbL@loop\bbL@tempa{##2}{% empty items and spaces are ok
1856     \advance\count@\@ne
1857     \toks@\expandafter{\bbL@tempa}%
1858     \bbL@exp{%
1859       \SetString\bbL@templ{\romannumeral\count@}{\the\toks@}%
1860       \count@=\the\count@\relax}}}%
1861 <</Macros local to BabelCommands>>

```

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

```

1862 \def\bbL@aftercmds#1{%
1863   \toks@\expandafter{\bbL@scafter#1}%
1864   \xdef\bbL@scafter{\the\toks@}

```

**Case mapping** The command `\SetCase` is deprecated. Currently it consists in a definition with a hack just for backward compatibility in the macro mapping.

```

1865 <<*Macros local to BabelCommands>> ≡
1866 \newcommand\SetCase[3][]{%
1867 \def\bbl@tempa####1####2{%
1868 \ifx####1\@empty\else
1869 \bbl@carg\bbl@add{extras\CurrentOption}{%
1870 \bbl@carg\babel@save{c__text_uppercase_\string####1_tl}%
1871 \bbl@carg\def{c__text_uppercase_\string####1_tl}{####2}%
1872 \bbl@carg\babel@save{c__text_lowercase_\string####2_tl}%
1873 \bbl@carg\def{c__text_lowercase_\string####2_tl}{####1}%
1874 \expandafter\bbl@tempa
1875 \fi}%
1876 \bbl@tempa##1\@empty\@empty
1877 \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1878 <</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.

```

1879 <<*Macros local to BabelCommands>> ≡
1880 \newcommand\SetHyphenMap[1]{%
1881 \bbl@forlang\bbl@tempa{%
1882 \expandafter\bbl@stringdef
1883 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1884 <</Macros local to BabelCommands>>

```

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

```

1885 \newcommand\BabelLower[2]{% one to one.
1886 \ifnum\lccode#1=#2\else
1887 \babel@savevariable{\lccode#1}%
1888 \lccode#1=#2\relax
1889 \fi}
1890 \newcommand\BabelLowerMM[4]{% many-to-many
1891 \@tempcnta=#1\relax
1892 \@tempcntb=#4\relax
1893 \def\bbl@tempa{%
1894 \ifnum\@tempcnta>#2\else
1895 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1896 \advance\@tempcnta#3\relax
1897 \advance\@tempcntb#3\relax
1898 \expandafter\bbl@tempa
1899 \fi}%
1900 \bbl@tempa}
1901 \newcommand\BabelLowerM0[4]{% many-to-one
1902 \@tempcnta=#1\relax
1903 \def\bbl@tempa{%
1904 \ifnum\@tempcnta>#2\else
1905 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1906 \advance\@tempcnta#3
1907 \expandafter\bbl@tempa
1908 \fi}%
1909 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1910 <<*More package options>> ≡
1911 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1912 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1913 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1914 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1915 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1916 <</More package options>>

```



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

```

1917 \AtEndOfPackage{%
1918   \ifx\bbbl@opt@hyphenmap\undefined
1919     \bbbl@xin@{,}\bbbl@language@opts}%
1920   \chardef\bbbl@opt@hyphenmap\ifin@4\else\one\fi
1921   \fi}

```

#### 4.14. Tailor captions

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.

```

1922 \newcommand\setlocalecaption{%%^A Catch typos.
1923   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1924 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1925   \bbbl@trim@def\bbbl@tempa{#2}%
1926   \bbbl@xin@{.template}\bbbl@tempa}%
1927   \ifin@
1928     \bbbl@ini@captions@template{#3}{#1}%
1929   \else
1930     \edef\bbbl@tempd{%
1931       \expandafter\expandafter\expandafter
1932       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1933     \bbbl@xin@
1934       {\expandafter\string\csname #2name\endcsname}%
1935       {\bbbl@tempd}%
1936     \ifin@ % Renew caption
1937       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
1938     \ifin@
1939       \bbbl@exp{%
1940         \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1941         {\\bbbl@scset\<#2name>\<#1#2name>}%
1942         {}}%
1943       \else % Old way converts to new way
1944         \bbbl@ifunset{#1#2name}%
1945         {\bbbl@exp{%
1946           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1947           \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1948           {\def\<#2name>\<#1#2name>}}%
1949           {}}}%
1950       {}%
1951     \fi
1952   \else
1953     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
1954     \ifin@ % New way
1955     \bbbl@exp{%
1956       \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}}%
1957       \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1958       {\\bbbl@scset\<#2name>\<#1#2name>}}%
1959       {}}%
1960     \else % Old way, but defined in the new way
1961     \bbbl@exp{%
1962       \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1963       \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1964       {\def\<#2name>\<#1#2name>}}%
1965       {}}%
1966     \fi%
1967   \fi
1968   \@namedef{#1#2name}{#3}%
1969   \toks@ \expandafter\bbbl@captionslist}%
1970   \bbbl@exp{\in@{\<#2name>}\the\toks@}}%
1971   \ifin@ \else
1972     \bbbl@exp{\bbbl@add\bbbl@captionslist{\<#2name>}}%

```

```

1973     \bbl@tglobal\bbl@captionslist
1974     \fi
1975     \fi}
1976 %^A \def\bbl@setcaption@s#1#2#3{} % Not yet implemented (w/o 'name')

```

## 4.15. 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 Tlenc.def.

**\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.

```

1977 \bbl@trace{Macros related to glyphs}
1978 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1979     \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
1980     \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.

```

1981 \def\save@sf@q#1{\leavevmode
1982     \begingroup
1983     \edef@SF{\spacefactor\the\spacefactor}#1@SF
1984     \endgroup}

```

### 4.15.1. 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.

```

1985 \ProvideTextCommand{\quotedblbase}{OT1}{%
1986     \save@sf@q{\set@low@box{\textquotedblright}/}%
1987     \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.

```

1988 \ProvideTextCommandDefault{\quotedblbase}{%
1989     \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1990 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1991     \save@sf@q{\set@low@box{\textquoteright}/}%
1992     \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.

```

1993 \ProvideTextCommandDefault{\quotesinglbase}{%
1994     \UseTextSymbol{OT1}{\quotesinglbase}}

```

**\guillemetleft**

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

```

1995 \ProvideTextCommand{\guillemetleft}{OT1}{%
1996     \ifmmode
1997         \ll
1998     \else
1999         \save@sf@q{\nobreak
2000             \raise.2ex\hbox{\scriptscriptstyle\ll}}\bbl@allowhyphens}%
2001     \fi}
2002 \ProvideTextCommand{\guillemetright}{OT1}{%
2003     \ifmmode
2004         \gg
2005     \else
2006         \save@sf@q{\nobreak

```

```

2007     \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
2008 \fi}
2009 \ProvideTextCommand{\guillemotleft}{OT1}{%
2010   \ifmode
2011     \ll
2012   \else
2013     \save@sf@q{\nobreak
2014       \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2015   \fi}
2016 \ProvideTextCommand{\guillemotright}{OT1}{%
2017   \ifmode
2018     \gg
2019   \else
2020     \save@sf@q{\nobreak
2021       \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
2022   \fi}

```

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

```

2023 \ProvideTextCommandDefault{\guillemetleft}{%
2024   \UseTextSymbol{OT1}{\guillemetleft}}
2025 \ProvideTextCommandDefault{\guillemetright}{%
2026   \UseTextSymbol{OT1}{\guillemetright}}
2027 \ProvideTextCommandDefault{\guillemotleft}{%
2028   \UseTextSymbol{OT1}{\guillemotleft}}
2029 \ProvideTextCommandDefault{\guillemotright}{%
2030   \UseTextSymbol{OT1}{\guillemotright}}

```

#### **\guilsinglleft**

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

```

2031 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2032   \ifmode
2033     <%
2034   \else
2035     \save@sf@q{\nobreak
2036       \raise.2ex\hbox{\scriptscriptstyle<}\bbl@allowhyphens}%
2037   \fi}
2038 \ProvideTextCommand{\guilsinglright}{OT1}{%
2039   \ifmode
2040     >%
2041   \else
2042     \save@sf@q{\nobreak
2043       \raise.2ex\hbox{\scriptscriptstyle>}\bbl@allowhyphens}%
2044   \fi}

```

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

```

2045 \ProvideTextCommandDefault{\guilsinglleft}{%
2046   \UseTextSymbol{OT1}{\guilsinglleft}}
2047 \ProvideTextCommandDefault{\guilsinglright}{%
2048   \UseTextSymbol{OT1}{\guilsinglright}}

```

### 4.15.2. Letters

#### **\ij**

**\IJ** The dutch language uses the letter ‘ij’. It is available in T1 encoded fonts, but not in the OT1 encoded fonts. Therefore we fake it for the OT1 encoding.

```

2049 \DeclareTextCommand{\ij}{OT1}{%
2050   i\kern-0.02em\bbl@allowhyphens j}
2051 \DeclareTextCommand{\IJ}{OT1}{%
2052   I\kern-0.02em\bbl@allowhyphens J}
2053 \DeclareTextCommand{\ij}{T1}{\char188}
2054 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2055 \ProvideTextCommandDefault{\ij}{%
2056 \UseTextSymbol{OT1}{\ij}}
2057 \ProvideTextCommandDefault{\IJ}{%
2058 \UseTextSymbol{OT1}{\IJ}}
```

**\dj**

**\DJ** The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```
2059 \def\crrtic@{\hrule height0.1ex width0.3em}
2060 \def\crttic@{\hrule height0.1ex width0.33em}
2061 \def\ddj@{%
2062 \setbox0\hbox{d}\dimen@=\ht0
2063 \advance\dimen@lex
2064 \dimen@.45\dimen@
2065 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2066 \advance\dimen@ii.5ex
2067 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2068 \def\DDJ@{%
2069 \setbox0\hbox{D}\dimen@=.55\ht0
2070 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2071 \advance\dimen@ii.15ex % correction for the dash position
2072 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2073 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2074 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2075 %
2076 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2077 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2078 \ProvideTextCommandDefault{\dj}{%
2079 \UseTextSymbol{OT1}{\dj}}
2080 \ProvideTextCommandDefault{\DJ}{%
2081 \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.

```
2082 \DeclareTextCommand{\SS}{OT1}{SS}
2083 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 4.15.3. 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**

**\grq** The ‘german’ single quotes.

```
2084 \ProvideTextCommandDefault{\glq}{%
2085 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2086 \ProvideTextCommand{\grq}{T1}{%
2087 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2088 \ProvideTextCommand{\grq}{TU}{%
2089 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2090 \ProvideTextCommand{\grq}{OT1}{%
2091 \save@sf@q{\kern-.0125em
2092 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%

```

```

2093 \kern.07em\relax}}
2094 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

### **\glqq**

**\grqq** The ‘german’ double quotes.

```

2095 \ProvideTextCommandDefault{\glqq}{%
2096 \textormath{\textquotedblbase}{\mbox{\textquotedblbase}}}

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

2097 \ProvideTextCommand{\grqq}{T1}{%
2098 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2099 \ProvideTextCommand{\grqq}{TU}{%
2100 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2101 \ProvideTextCommand{\grqq}{0T1}{%
2102 \save@sf@q{\kern-.07em
2103 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2104 \kern.07em\relax}}
2105 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

### **\flq**

**\frq** The ‘french’ single guillemets.

```

2106 \ProvideTextCommandDefault{\flq}{%
2107 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2108 \ProvideTextCommandDefault{\frq}{%
2109 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

### **\flqq**

**\frqq** The ‘french’ double guillemets.

```

2110 \ProvideTextCommandDefault{\flqq}{%
2111 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2112 \ProvideTextCommandDefault{\frqq}{%
2113 \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

## 4.15.4. 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**

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

```

2114 \def\umlauthigh{%
2115 \def\bbl@umlauta##1{\leavevmode\bgroup%
2116 \accent\csname\f@encoding dqpos\endcsname
2117 ##1\bbl@allowhyphens\egroup}%
2118 \let\bbl@umlaute\bbl@umlauta}
2119 \def\umlautlow{%
2120 \def\bbl@umlauta{\protect\lower@umlaut}}
2121 \def\umlautelow{%
2122 \def\bbl@umlaute{\protect\lower@umlaut}}
2123 \umlauthigh

```

**\lower@umlaut** 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.

```
2124 \expandafter\ifx\csname U@D\endcsname\relax
2125 \csname newdimen\endcsname\U@D
2126 \fi
```

The following code fools T<sub>E</sub>X's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```
2127 \def\lower@umlaut#1{%
2128 \leavevmode\bgroup
2129 \U@D lex%
2130 {\setbox\z@\hbox{%
2131 \char\csname\fontencoding dqpos\endcsname}%
2132 \dimen@ -.45ex\advance\dimen@ht\z@
2133 \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2134 \accent\csname\fontencoding dqpos\endcsname
2135 \fontdimen5\font\U@D #1%
2136 \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).

```
2137 \AtBeginDocument{%
2138 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2139 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2140 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2141 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
2142 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2143 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2144 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2145 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2146 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2147 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2148 \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.

```
2149 \ifx\l@english\@undefined
2150 \chardef\l@english\z@
2151 \fi
2152 % The following is used to cancel rules in ini files (see Amharic).
2153 \ifx\l@unhyphenated\@undefined
2154 \newlanguage\l@unhyphenated
2155 \fi
```

## 4.16. Layout

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

```
2156 \bbl@trace{Bidi layout}
2157 \providecommand\IfBabelLayout[3]{#3}%
```

## 4.17. Load engine specific macros

Some macros are not defined in all engines, so, after loading the files define them if necessary to raise an error.

```
2158 \bbl@trace{Input engine specific macros}
2159 \ifcase\bbl@engine
2160   \input txtbabel.def
2161 \or
2162   \input luababel.def
2163 \or
2164   \input xebabel.def
2165 \fi
2166 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}
2167 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}
2168 \ifx\babelposthyphenation\undefined
2169   \let\babelposthyphenation\babelprehyphenation
2170   \let\babelpatterns\babelprehyphenation
2171   \let\babelcharproperty\babelprehyphenation
2172 \fi
2173 </package | core>
```

## 4.18. Creating and modifying languages

Continue with  $\LaTeX$  only.

`\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.

```
2174 <{*package}
2175 \bbl@trace{Creating languages and reading ini files}
2176 \let\bbl@extend@ini@gobble
2177 \newcommand\babelprovide[2][{}]{%
2178   \let\bbl@save@langname\languagename
2179   \edef\bbl@savelocaleid{\the\localeid}%
2180   % Set name and locale id
2181   \edef\languagename{#2}%
2182   \bbl@id@assign
2183   % Initialize keys
2184   \bbl@vforeach{captions,date,import,main,script,language,%
2185     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2186     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2187     Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2188     {\bbl@csarg\let{KVP@##1}\@nnil}%
2189   \global\let\bbl@release@transforms@empty
2190   \global\let\bbl@release@casing@empty
2191   \let\bbl@calendars@empty
2192   \global\let\bbl@inidata@empty
2193   \global\let\bbl@extend@ini@gobble
2194   \global\let\bbl@included@inis@empty
2195   \gdef\bbl@key@list{;}%
2196   \bbl@ifunset{bbl@passto@#2}%
2197     {\def\bbl@tempa{#1}}%
2198     {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}}%
2199   \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2200     \in@/{/#1}% With /, (re)sets a value in the ini
2201     \ifin@
2202       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2203       \bbl@renewinikey##1@@{##2}%
2204     \else
2205       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2206         \bbl@error{unknown-provide-key}{#1}{}%
2207       \fi
2208       \bbl@csarg\def{KVP@##1}{##2}%
2209     \fi}%
```

```

2210 \chardef\bb@l@howloaded=% 0:none; 1:ldf without ini; 2:ini
2211 \bb@ifunset{date#2}\z@{\bb@ifunset{bb@llevel@#2}\one\tw@}%
2212 % == init ==
2213 \ifx\bb@l@screset\@undefined
2214 \bb@ldfinit
2215 \fi
2216 % ==
2217 \ifx\bb@l@KVP@import\@nnil\else \ifx\bb@l@KVP@import\@nnil
2218 \def\bb@l@KVP@import{\@empty}%
2219 \fi\fi
2220 % == date (as option) ==
2221 % \ifx\bb@l@KVP@date\@nnil\else
2222 % \fi
2223 % ==
2224 \let\bb@l@l@bkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2225 \ifcase\bb@l@howloaded
2226 \let\bb@l@l@bkflag\@empty % new
2227 \else
2228 \ifx\bb@l@KVP@hyphenrules\@nnil\else
2229 \let\bb@l@l@bkflag\@empty
2230 \fi
2231 \ifx\bb@l@KVP@import\@nnil\else
2232 \let\bb@l@l@bkflag\@empty
2233 \fi
2234 \fi
2235 % == import, captions ==
2236 \ifx\bb@l@KVP@import\@nnil\else
2237 \bb@exp{\@empty\bb@ifblank{\bb@l@KVP@import}}%
2238 {\ifx\bb@l@initload\relax
2239 \begingroup
2240 \def\BabelBeforeIni##1##2{\gdef\bb@l@KVP@import{##1}\endinput}%
2241 \bb@input@texini{##2}%
2242 \endgroup
2243 \else
2244 \xdef\bb@l@KVP@import{\bb@l@initload}%
2245 \fi}%
2246 {}%
2247 \let\bb@l@KVP@date\@empty
2248 \fi
2249 \let\bb@l@KVP@captions@\bb@l@KVP@captions
2250 \ifx\bb@l@KVP@captions\@nnil
2251 \let\bb@l@KVP@captions\bb@l@KVP@import
2252 \fi
2253 % ==
2254 \ifx\bb@l@KVP@transforms\@nnil\else
2255 \bb@replace\bb@l@KVP@transforms{ },}%
2256 \fi
2257 % == Load ini ==
2258 \ifcase\bb@l@howloaded
2259 \bb@provide@new{##2}%
2260 \else
2261 \bb@ifblank{##1}%
2262 {}% With \bb@load@basic below
2263 {\bb@provide@renew{##2}}%
2264 \fi
2265 % == include == TODO
2266 % \ifx\bb@l@included@inis\@empty\else
2267 % \bb@replace\bb@l@included@inis{ },}%
2268 % \bb@foreach\bb@l@included@inis{%
2269 % \openin\bb@l@readstream=babel-##1.ini
2270 % \bb@extend@ini{##2}}%
2271 % \closein\bb@l@readstream
2272 % \fi

```



```

2273 % Post tasks
2274 % -----
2275 % == subsequent calls after the first provide for a locale ==
2276 \ifx\bbbl@inidata\@empty\else
2277   \bbbl@extend@ini{#2}%
2278 \fi
2279 % == ensure captions ==
2280 \ifx\bbbl@KVP@captions\@nnil\else
2281   \bbbl@ifunset{bbbl@extracaps@#2}%
2282     {\bbbl@exp{\\\babelensure[exclude=\\today]{#2}}}%
2283     {\bbbl@exp{\\\babelensure[exclude=\\today,
2284               include=\[bbbl@extracaps@#2]]{#2}}}%
2285   \bbbl@ifunset{bbbl@ensure@language}%
2286     {\bbbl@exp{%
2287       \\DeclareRobustCommand\<bbbl@ensure@language>[1]{%
2288         \\foreignlanguage{language}%
2289           {###1}}}}%
2290     {}%
2291   \bbbl@exp{%
2292     \\bbbl@tglobal\<bbbl@ensure@language>%
2293     \\bbbl@tglobal\<bbbl@ensure@language\space>}%
2294 \fi

```

At this point all parameters are defined if 'import'. Now we execute some code depending on them. But what about if nothing was imported? We just set the basic parameters, but still loading the whole ini file.

```

2295 \bbbl@load@basic{#2}%
2296 % == script, language ==
2297 % Override the values from ini or defines them
2298 \ifx\bbbl@KVP@script\@nnil\else
2299   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2300 \fi
2301 \ifx\bbbl@KVP@language\@nnil\else
2302   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2303 \fi
2304 \ifcase\bbbl@engine\or
2305   \bbbl@ifunset{bbbl@chrng@language}{}%
2306   {\directlua{
2307     Babel.set_chANGES_b('\bbbl@cl{sbcpr}', '\bbbl@cl{chrng}') }}%
2308 \fi
2309 % == Line breaking: intraspaces, intrapenalty ==
2310 % For CJK, East Asian, Southeast Asian, if interspace in ini
2311 \ifx\bbbl@KVP@intraspace\@nnil\else % We can override the ini or set
2312   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2313 \fi
2314 \bbbl@provide@intraspace
2315 % == Line breaking: justification ==
2316 \ifx\bbbl@KVP@justification\@nnil\else
2317   \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2318 \fi
2319 \ifx\bbbl@KVP@linebreaking\@nnil\else
2320   \bbbl@xin@{,\bbbl@KVP@linebreaking,}%
2321   {,elongated,kashida,cjk,padding,unhyphenated,}%
2322   \ifin@
2323     \bbbl@csarg\xdef
2324       {lnbrk@language}{\expandafter\car\bbbl@KVP@linebreaking\@nil}%
2325   \fi
2326 \fi
2327 \bbbl@xin@{/e}{/\bbbl@cl{lnbrk}}%
2328 \ifin@\else\bbbl@xin@{/k}{/\bbbl@cl{lnbrk}}\fi
2329 \ifin@\bbbl@arabicjust\fi
2330 % WIP
2331 \bbbl@xin@{/p}{/\bbbl@cl{lnbrk}}%

```

```

2332 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2333 % == Line breaking: hyphenate.other.(locale|script) ==
2334 \ifx\bbl@lbfkflag@empty
2335   \bbl@ifunset{bbl@hyotl@languagename}{}%
2336   {\bbl@csarg\bbl@replace{hyotl@languagename}{ }{,}%
2337   \bbl@startcommands*{languagename}{}%
2338   \bbl@csarg\bbl@foreach{hyotl@languagename}{%
2339     \ifcase\bbl@engine
2340     \ifnum##1<257
2341       \SetHyphenMap{\BabelLower{##1}{##1}}%
2342     \fi
2343     \else
2344       \SetHyphenMap{\BabelLower{##1}{##1}}%
2345     \fi}%
2346   \bbl@endcommands}%
2347 \bbl@ifunset{bbl@hyots@languagename}{}%
2348 {\bbl@csarg\bbl@replace{hyots@languagename}{ }{,}%
2349 \bbl@csarg\bbl@foreach{hyots@languagename}{%
2350   \ifcase\bbl@engine
2351   \ifnum##1<257
2352     \global\lccode##1=##1\relax
2353   \fi
2354   \else
2355     \global\lccode##1=##1\relax
2356   \fi}}%
2357 \fi
2358 % == Counters: maparabic ==
2359 % Native digits, if provided in ini (TeX level, xe and lua)
2360 \ifcase\bbl@engine\else
2361   \bbl@ifunset{bbl@dgnat@languagename}{}%
2362   {\expandafter\ifx\csname bbl@dgnat@languagename\endcsname\@empty\else
2363   \expandafter\expandafter\expandafter
2364   \bbl@setdigits\csname bbl@dgnat@languagename\endcsname
2365   \ifx\bbl@KVP@maparabic\@nnil\else
2366   \ifx\bbl@latinarabic\@undefined
2367   \expandafter\let\expandafter\@arabic
2368   \csname bbl@counter@languagename\endcsname
2369   \else % i.e., if layout=counters, which redefines \@arabic
2370   \expandafter\let\expandafter\bbl@latinarabic
2371   \csname bbl@counter@languagename\endcsname
2372   \fi
2373   \fi
2374   \fi}%
2375 \fi
2376 % == Counters: mapdigits ==
2377 % > luabel.def
2378 % == Counters: alph, Alph ==
2379 \ifx\bbl@KVP@alph\@nnil\else
2380   \bbl@exp{%
2381     \\bbl@add\<bbl@preextras@languagename>{%
2382     \\babel@save\\@alph
2383     \let\\@alph\<bbl@cntr@bbl@KVP@alph @languagename>}}%
2384 \fi
2385 \ifx\bbl@KVP@Alph\@nnil\else
2386   \bbl@exp{%
2387     \\bbl@add\<bbl@preextras@languagename>{%
2388     \\babel@save\\@Alph
2389     \let\\@Alph\<bbl@cntr@bbl@KVP@Alph @languagename>}}%
2390 \fi
2391 % == Casing ==
2392 \bbl@release@casing
2393 \ifx\bbl@KVP@casing\@nnil\else
2394   \bbl@csarg\xdef{casing@languagename}%

```

```

2395     {\nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2396 \fi
2397 % == Calendars ==
2398 \ifx\bbl@KVP@calendar\@nnil
2399   \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2400 \fi
2401 \def\bbl@tempe##1 ##2\@{% Get first calendar
2402   \def\bbl@tempa{##1}%
2403   \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@}%
2404 \def\bbl@tempe##1.##2.##3\@{%
2405   \def\bbl@tempc{##1}%
2406   \def\bbl@tempb{##2}%
2407 \expandafter\bbl@tempe\bbl@tempa.\@
2408 \bbl@csarg\edef{calpr@\languagename}{%
2409   \ifx\bbl@tempc@empty\else
2410     calendar=\bbl@tempc
2411   \fi
2412   \ifx\bbl@tempb@empty\else
2413     ,variant=\bbl@tempb
2414   \fi}%
2415 % == engine specific extensions ==
2416 % Defined in XXXbabel.def
2417 \bbl@provide@extra{##2}%
2418 % == require.babel in ini ==
2419 % To load or reload the babel-*.tex, if require.babel in ini
2420 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2421   \bbl@ifunset{bbl@rqtex@\languagename}{}%
2422   {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2423     \let\BabelBeforeIni@gobbletwo
2424     \chardef\atcatcode=\catcode \@
2425     \catcode \@=11\relax
2426     \def\CurrentOption{##2}%
2427     \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2428     \catcode \@=\atcatcode
2429     \let\atcatcode\relax
2430     \global\bbl@csarg\let{rqtex@\languagename}\relax
2431   \fi}%
2432 \bbl@foreach\bbl@calendars{%
2433   \bbl@ifunset{bbl@ca##1}{%
2434     \chardef\atcatcode=\catcode \@
2435     \catcode \@=11\relax
2436     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2437     \catcode \@=\atcatcode
2438     \let\atcatcode\relax}%
2439   {}}%
2440 \fi
2441 % == frenchspacing ==
2442 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2443 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2444 \ifin@
2445   \bbl@extras@wrap{\bbl@pre@fs}%
2446   {\bbl@pre@fs}%
2447   {\bbl@post@fs}%
2448 \fi
2449 % == transforms ==
2450 % > luababel.def
2451 \def\CurrentOption{##2}%
2452 \@nameuse{bbl@icsave@##2}%
2453 % == main ==
2454 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2455   \let\languagename\bbl@savelangname
2456   \chardef\localeid\bbl@savelocaleid\relax
2457 \fi

```

```

2458 % == hyphenrules (apply if current) ==
2459 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2460   \ifnum\bbbl@savelocaleid=\localeid
2461     \language\@nameuse{l@\languagename}%
2462   \fi
2463 \fi}

```

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

```

2464 \def\bbbl@provide@new#1{%
2465   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2466   \@namedef{extras#1}{}%
2467   \@namedef{noextras#1}{}%
2468   \bbbl@startcommands*{#1}{captions}%
2469   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2470     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2471       \ifx##1\@nnil\else
2472         \bbbl@exp{%
2473           \\SetString\\##1{%
2474             \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2475           \expandafter\bbbl@tempb
2476         \fi}%
2477     \expandafter\bbbl@tempb\bbbl@captionslist\@nnil
2478   \else
2479     \ifx\bbbl@initoload\relax
2480       \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2481     \else
2482       \bbbl@read@ini{\bbbl@initoload}2% % Same
2483     \fi
2484   \fi
2485   \StartBabelCommands*{#1}{date}%
2486   \ifx\bbbl@KVP@date\@nnil
2487     \bbbl@exp{%
2488       \\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2489   \else
2490     \bbbl@savetoday
2491     \bbbl@savedate
2492   \fi
2493   \bbbl@endcommands
2494   \bbbl@load@basic{#1}%
2495   % == hyphenmins == (only if new)
2496   \bbbl@exp{%
2497     \gdef\<#1hyphenmins>{%
2498       {\bbbl@ifunset{\bbbl@lfthm#1}{2}{\bbbl@cs{lfthm#1}}}%
2499       {\bbbl@ifunset{\bbbl@rgthm#1}{3}{\bbbl@cs{rgthm#1}}}}}%
2500   % == hyphenrules (also in renew) ==
2501   \bbbl@provide@hyphens{#1}%
2502   \ifx\bbbl@KVP@main\@nnil\else
2503     \expandafter\main@language\expandafter{#1}%
2504   \fi}
2505 %
2506 \def\bbbl@provide@renew#1{%
2507   \ifx\bbbl@KVP@captions\@nnil\else
2508     \StartBabelCommands*{#1}{captions}%
2509     \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2510     \EndBabelCommands
2511   \fi
2512   \ifx\bbbl@KVP@date\@nnil\else
2513     \StartBabelCommands*{#1}{date}%
2514     \bbbl@savetoday
2515     \bbbl@savedate
2516     \EndBabelCommands
2517   \fi

```

```

2518 % == hyphenrules (also in new) ==
2519 \ifx\bbbl@l@bkflag\@empty
2520   \bbbl@provide@hyphens{#1}%
2521 \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.

```

2522 \def\bbbl@load@basic#1{%
2523   \ifcase\bbbl@howloaded\or\or
2524     \ifcase\csname bbl@llevel@\languagename\endcsname
2525       \bbbl@csarg\let{lname@\languagename}\relax
2526     \fi
2527   \fi
2528   \bbbl@ifunset{bbbl@lname@#1}%
2529   {\def\BabelBeforeIni##1##2{%
2530     \begingroup
2531       \let\bbbl@ini@captions@aux\@gobbletwo
2532       \def\bbbl@inidate ###1.###2.###3.###4\relax ###5###6}%
2533     \bbbl@read@ini{##1}l%
2534     \ifx\bbbl@initoload\relax\endinput\fi
2535     \endgroup}%
2536   \begingroup      % boxed, to avoid extra spaces:
2537     \ifx\bbbl@initoload\relax
2538       \bbbl@input@texini{#1}%
2539     \else
2540       \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}{}}%
2541     \fi
2542   \endgroup}%
2543   {}}

```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```

2544 \def\bbbl@provide@hyphens#1{%
2545   \@tempcnta\m@ne % a flag
2546   \ifx\bbbl@KVP@hyphenrules\@nnil\else
2547     \bbbl@replace\bbbl@KVP@hyphenrules{ },}%
2548     \bbbl@foreach\bbbl@KVP@hyphenrules{%
2549       \ifnum\@tempcnta=\m@ne % if not yet found
2550         \bbbl@ifsamestring{##1}{+}%
2551         {\bbbl@carg\addlanguage{l@##1}}%
2552         {}%
2553         \bbbl@ifunset{l@##1}% After a possible +
2554         {}%
2555         {\@tempcnta\@nameuse{l@##1}}%
2556       \fi}%
2557   \ifnum\@tempcnta=\m@ne
2558     \bbbl@warning{%
2559       Requested 'hyphenrules' for '\languagename' not found:\\%
2560       \bbbl@KVP@hyphenrules.\\%
2561       Using the default value. Reported}%
2562   \fi
2563 \fi
2564 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2565   \ifx\bbbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2566     \bbbl@ifunset{bbbl@hyphr@#1}{}% use value in ini, if exists
2567     {\bbbl@exp{\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
2568     {}%
2569     {\bbbl@ifunset{l@\bbbl@cl{hyphr}}}%
2570     {}% if hyphenrules found:
2571     {\@tempcnta\@nameuse{l@\bbbl@cl{hyphr}}}}%
2572   \fi
2573 \fi
2574 \bbbl@ifunset{l@#1}%

```

```

2575     {\ifnum \@tempcnta=\m@ne
2576       \bbl@carg\adddialect{l@#1}\language
2577     \else
2578       \bbl@carg\adddialect{l@#1}\@tempcnta
2579     \fi}%
2580 {\ifnum \@tempcnta=\m@ne\else
2581   \global\bbl@carg\chardef{l@#1}\@tempcnta
2582   \fi}}

```

The reader of babel - . . . tex files. We reset temporarily some catcodes (and make sure no space is accidentally inserted).

```

2583 \def\bbl@input@texini#1{%
2584   \bbl@bsphack
2585   \bbl@exp{%
2586     \catcode`\\%=14 \catcode`\\%=0
2587     \catcode`\\={1 \catcode`\\}=2
2588     \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2589     \catcode`\\%=the\catcode`%\relax
2590     \catcode`\\%=the\catcode`\\relax
2591     \catcode`\\={the\catcode`{\relax
2592     \catcode`\\}=the\catcode`}\relax}%
2593   \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.

```

2594 \def\bbl@iniline#1\bbl@iniline{%
2595   \ifnextchar[\bbl@inisect{\ifnextchar;\bbl@iniskip\bbl@inistore}#1@@}% ]
2596 \def\bbl@inisect[#1]#2@@{\def\bbl@section{#1}}
2597 \def\bbl@iniskip#1\@{\%      if starts with ;
2598 \def\bbl@inistore#1=#2\@{\%    full (default)
2599   \bbl@trim@def\bbl@tempa{#1}%
2600   \bbl@trim\toks@{#2}%
2601   \bbl@ifsamestring{\bbl@tempa}{\include}%
2602   {\bbl@read@subini{\the\toks@}}%
2603   {\bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2604   \ifin@else
2605     \bbl@xin@{,identification/include.}%
2606     {,\bbl@section/\bbl@tempa}%
2607     \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2608     \bbl@exp{%
2609       \\g@addto@macro\\bbl@inidata{%
2610         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2611     \fi}}
2612 \def\bbl@inistore@min#1=#2\@{\%  minimal (maybe set in \bbl@read@ini)
2613   \bbl@trim@def\bbl@tempa{#1}%
2614   \bbl@trim\toks@{#2}%
2615   \bbl@xin@{.identification.}{.\bbl@section.}%
2616   \ifin@
2617     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2618       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2619   \fi}

```

## 4.19. Main loop in 'provide'

Now, the 'main loop', \bbl@read@ini, 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.

\bbl@loop@ini is the reader, line by line (1: stream), and calls \bbl@iniline to save the key/value pairs. If \bbl@inistore finds the @include directive, the input stream is switched temporarily and \bbl@read@subini is called.

```

2620 \def\bbl@loop@ini#1{%
2621   \loop
2622     \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2623     \endlinechar@m@ne
2624     \read#1 to \bbl@line
2625     \endlinechar`\^^M
2626     \ifx\bbl@line\empty\else
2627       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2628     \fi
2629   \repeat}
2630 \def\bbl@read@subini#1{%
2631   \ifx\bbl@readsubstream\undefined
2632     \csname newread\endcsname\bbl@readsubstream
2633   \fi
2634   \openin\bbl@readsubstream=babel-#1.ini
2635   \ifeof\bbl@readsubstream
2636     \bbl@error{no-ini-file}{#1}{}%
2637   \else
2638     {\bbl@loop@ini\bbl@readsubstream}%
2639   \fi
2640   \closein\bbl@readsubstream}
2641 \ifx\bbl@readstream\undefined
2642   \csname newread\endcsname\bbl@readstream
2643 \fi
2644 \def\bbl@read@ini#1#2{%
2645   \global\let\bbl@extend@ini@gobble
2646   \openin\bbl@readstream=babel-#1.ini
2647   \ifeof\bbl@readstream
2648     \bbl@error{no-ini-file}{#1}{}%
2649   \else
2650     % == Store ini data in \bbl@inidata ==
2651     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2652     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2653     \bbl@info{Importing
2654               \ifcase#2font and identification \or basic \fi
2655               data for \languagename\%
2656               from babel-#1.ini. Reported}%
2657     \ifnum#2=\z@
2658       \global\let\bbl@inidata\empty
2659       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2660     \fi
2661     \def\bbl@section{identification}%
2662     \bbl@exp{\bbl@inistore tag.ini=#1\@@}%
2663     \bbl@inistore load.level=#2\@@
2664     \bbl@loop@ini\bbl@readstream
2665     % == Process stored data ==
2666     \bbl@csarg\xdef{lini@\languagename}{#1}%
2667     \bbl@read@ini@aux
2668     % == 'Export' data ==
2669     \bbl@ini@exports{#2}%
2670     \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2671     \global\let\bbl@inidata\empty
2672     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
2673     \bbl@tglobal\bbl@ini@loaded
2674   \fi
2675   \closein\bbl@readstream}
2676 \def\bbl@read@ini@aux{%
2677   \let\bbl@savestrings\empty
2678   \let\bbl@savetoday\empty
2679   \let\bbl@savestate\empty
2680   \def\bbl@elt##1##2##3{%
2681     \def\bbl@section{##1}%
2682     \in@{=date.}{=##1}% Find a better place

```

```

2683 \ifin@
2684 \bbl@ifunset{bbl@inikv@##1}%
2685 {\bbl@ini@calendar{##1}}%
2686 }%
2687 \fi
2688 \bbl@ifunset{bbl@inikv@##1}{}%
2689 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2690 \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.

```

2691 \def\bbl@extend@ini@aux#1{%
2692 \bbl@startcommands*{#1}{captions}%
2693 % Activate captions/... and modify exports
2694 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2695 \setlocalecaption{#1}{##1}{##2}}%
2696 \def\bbl@inikv@captions##1##2{%
2697 \bbl@ini@captions@aux{##1}{##2}}%
2698 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2699 \def\bbl@exportkey##1##2##3{%
2700 \bbl@ifunset{bbl@kv@##2}{%
2701 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2702 \bbl@exp{\global\let<bbl@##1@\languagename>\<bbl@kv@##2>}}%
2703 \fi}}%
2704 % As with \bbl@read@ini, but with some changes
2705 \bbl@read@ini@aux
2706 \bbl@ini@exports\tw@
2707 % Update inidata@lang by pretending the ini is read.
2708 \def\bbl@elt##1##2##3{%
2709 \def\bbl@section{##1}%
2710 \bbl@iniline##2=##3\bbl@iniline}%
2711 \csname bbl@inidata@#1\endcsname
2712 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2713 \StartBabelCommands*{#1}{date}% And from the import stuff
2714 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2715 \bbl@savetoday
2716 \bbl@savedate
2717 \bbl@endcommands}

```

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

```

2718 \def\bbl@ini@calendar#1{%
2719 \lowercase{\def\bbl@tempa{=#1=}}%
2720 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2721 \bbl@replace\bbl@tempa{=date.}{}%
2722 \in@{.licr=}{#1=}%
2723 \ifin@
2724 \ifcase\bbl@engine
2725 \bbl@replace\bbl@tempa{.licr=}{}%
2726 \else
2727 \let\bbl@tempa\relax
2728 \fi
2729 \fi
2730 \ifx\bbl@tempa\relax\else
2731 \bbl@replace\bbl@tempa{=}{}%
2732 \ifx\bbl@tempa\@empty\else
2733 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2734 \fi
2735 \bbl@exp{%
2736 \def<bbl@inikv@#1>####1####2{%
2737 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}%
2738 \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).

```

2739 \def\bbl@renewinikey#1/#2\@#3{%
2740 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2741 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2742 \bbl@trim\toks@{#3}% value
2743 \bbl@exp{%
2744 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2745 \\g@addto@macro\\bbl@inidata{%
2746 \\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.

```

2747 \def\bbl@exportkey#1#2#3{%
2748 \bbl@ifunset{\bbl@kv@#2}%
2749 {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2750 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2751 \bbl@csarg\gdef{#1@\languagename}{#3}}%
2752 \else
2753 \bbl@exp{\global\let<bbl@#1@\languagename>\<bbl@kv@#2>}%
2754 \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.

Although BCP 47 doesn't treat '-x-' as an extension, the CLDR and many other sources do (as a *private use extension*). For consistency with other single-letter subtags or 'singletons', here is considered an extension, too.

The identification section is used internally by babel in the following places [to be completed]: BCP 47 script tag in the Unicode ranges, which is in turn used by onchar; the language system is set with the names, and then fontspec maps them to the opentype tags, but if the latter package doesn't define them, then babel does it; encodings are used in pdftex to select a font encoding valid (and preloaded) for a language loaded on the fly.

```

2755 \def\bbl@iniwarning#1{%
2756 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2757 {\bbl@warning{%
2758 From babel-\bbl@cs{lini@\languagename}.ini:\\%
2759 \bbl@cs{@kv@identification.warning#1}\\%
2760 Reported }}}
2761 %
2762 \let\bbl@release@transforms\@empty
2763 \let\bbl@release@casing\@empty
2764 \def\bbl@ini@exports#1{%
2765 % Identification always exported
2766 \bbl@iniwarning{}%
2767 \ifcase\bbl@engine
2768 \bbl@iniwarning{.pdflatex}%
2769 \or
2770 \bbl@iniwarning{.lualatex}%
2771 \or
2772 \bbl@iniwarning{.xelatex}%
2773 \fi%
2774 \bbl@exportkey{lllevel}{identification.load.level}{}%
2775 \bbl@exportkey{elname}{identification.name.english}{}%
2776 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2777 {\csname bbl@elname@\languagename\endcsname}}%
2778 \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2779 % Somewhat hackish. TODO:
2780 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2781 \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2782 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2783 \bbl@exportkey{esname}{identification.script.name}{}%

```

```

2784 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2785   {\csname bbl@esname@\languagename\endcsname}}%
2786 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2787 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2788 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2789 \bbl@exportkey{vbc}{identification.variant.tag.bcp47}{}%
2790 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2791 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2792 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2793 % Also maps bcp47 -> languagename
2794 \ifbbl@bcptoname
2795   \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\languagename}%
2796 \fi
2797 \ifcase\bbl@engine\or
2798   \directlua{%
2799     Babel.locale_props[\the\bbl@cs{id@\languagename}].script
2800     = '\bbl@cl{sbc}}'%
2801 \fi
2802 % Conditional
2803 \ifnum#1>\z@      % 0 = only info, 1, 2 = basic, (re)new
2804   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2805   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2806   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2807   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2808   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2809   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2810   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2811   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2812   \bbl@exportkey{intsp}{typography.intraspaces}{}%
2813   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2814   \bbl@exportkey{chrng}{characters.ranges}{}%
2815   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2816   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2817   \ifnum#1=\tw@      % only (re)new
2818     \bbl@exportkey{rqtex}{identification.require.babel}{}%
2819     \bbl@tglobal\bbl@savetoday
2820     \bbl@tglobal\bbl@savestate
2821     \bbl@savestrings
2822   \fi
2823 \fi}

```

## 4.20. Processing keys in ini

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

```

2824 \def\bbl@inikv#1#2{%      key=value
2825   \toks@{#2}%             This hides #'s from ini values
2826   \bbl@csarg\edef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2827 \let\bbl@inikv@identification\bbl@inikv
2828 \let\bbl@inikv@date\bbl@inikv
2829 \let\bbl@inikv@typography\bbl@inikv
2830 \let\bbl@inikv@numbers\bbl@inikv

```

The characters section also stores the values, but casing is treated in a different fashion. Much like transforms, a set of commands calling the parser are stored in `\bbl@release@casing`, which is executed in `\babelprovide`.

```

2831 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2832 \def\bbl@inikv@characters#1#2{%
2833   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2834   {\bbl@exp{%
2835     \\g@addto@macro\\bbl@release@casing{%
2836       \\bbl@casemapping}{\languagename}{\unexpanded{#2}}}}%

```

```

2837 {\in@{${casing.}{#1}% e.g., casing.Uv = uV
2838 \ifin@
2839 \lowercase{\def\bb@tempb{#1}}%
2840 \bb@replace\bb@tempb{casing.}{}%
2841 \bb@exp{\g@addto@macro\bb@release@casing{%
2842 \bb@casemapping
2843 {\bb@maybextx\bb@tempb}{\languagename}{\unexpanded{#2}}}%
2844 \else
2845 \bb@inikv{#1}{#2}%
2846 \fi}}

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localenumeral, and another one preserving the trailing .1 for the ‘units’.

```

2847 \def\bb@inikv@counters#1#2{%
2848 \bb@ifsamestring{#1}{digits}%
2849 {\bb@error{digits-is-reserved}{}}}%
2850 {}%
2851 \def\bb@tempc{#1}%
2852 \bb@trim@def{\bb@tempb*}{#2}%
2853 \in@{.1$}{#1$}%
2854 \ifin@
2855 \bb@replace\bb@tempc{.1}{}%
2856 \bb@csarg\protected@xdef{cnt@\bb@tempc @\languagename}{%
2857 \noexpand\bb@alphanumeric{\bb@tempc}}%
2858 \fi
2859 \in@{.F.}{#1}%
2860 \ifin@else\in@{.S.}{#1}\fi
2861 \ifin@
2862 \bb@csarg\protected@xdef{cnt@#1@\languagename}{\bb@tempb*}%
2863 \else
2864 \toks@{}% Required by \bb@buildifcase, which returns \bb@tempa
2865 \expandafter\bb@buildifcase\bb@tempb* \ \ % Space after \
2866 \bb@csarg{\global\expandafter\let}{cnt@#1@\languagename}\bb@tempa
2867 \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.

```

2868 \ifcase\bb@engine
2869 \bb@csarg\def{inikv@captions.licr}#1#2{%
2870 \bb@ini@captions@aux{#1}{#2}}
2871 \else
2872 \def\bb@inikv@captions#1#2{%
2873 \bb@ini@captions@aux{#1}{#2}}
2874 \fi

```

The auxiliary macro for captions define \langlecaption>name.

```

2875 \def\bb@ini@captions@template#1#2{% string language tempa=capt-name
2876 \bb@replace\bb@tempa{.template}{}%
2877 \def\bb@toreplace{#1}{}%
2878 \bb@replace\bb@toreplace{[ ]}{\nobreakspace}}%
2879 \bb@replace\bb@toreplace{[ ]}{\csname}%
2880 \bb@replace\bb@toreplace{[ ]}{\csname the}%
2881 \bb@replace\bb@toreplace{[ ]}{\name\endcsname}}%
2882 \bb@replace\bb@toreplace{[ ]}{\endcsname}}%
2883 \bb@xin@{,\bb@tempa,}{,chapter,appendix,part,}%
2884 \ifin@
2885 \@nameuse{\bb@patch\bb@tempa}%
2886 \global\bb@csarg\let{\bb@tempa fmt@#2}\bb@toreplace
2887 \fi
2888 \bb@xin@{,\bb@tempa,}{,figure,table,}%
2889 \ifin@
2890 \global\bb@csarg\let{\bb@tempa fmt@#2}\bb@toreplace

```

```

2891 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2892   \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\language}%
2893   {\[fnum@\bbl@tempa]}%
2894   {\\\@nameuse{bbl@\bbl@tempa fmt@\\language}}}}%
2895 \fi}
2896 \def\bbl@ini@captions@aux#1#2{%
2897   \bbl@trim@def\bbl@tempa{#1}%
2898   \bbl@xin@{.template}{\bbl@tempa}%
2899   \ifin@
2900     \bbl@ini@captions@template{#2}\language
2901   \else
2902     \bbl@ifblank{#2}%
2903     {\bbl@exp{%
2904       \toks@{\\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2905       {\bbl@trim\toks@{#2}}}%
2906     \bbl@exp{%
2907       \\\bbl@add\\bbl@savestrings{%
2908         \\\SetString\<\bbl@tempa name>{\the\toks@}}%
2909       \toks@expandafter{\bbl@captionslist}%
2910       \bbl@exp{\\\in{\<\bbl@tempa name>}{\the\toks@}}%
2911       \ifin\else
2912         \bbl@exp{%
2913           \\\bbl@add\<bbl@extracaps@language>{\<\bbl@tempa name>}%
2914           \\\bbl@tglobal\<bbl@extracaps@language>}%
2915         \fi
2916       \fi}

```

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

```

2917 \def\bbl@list@the{%
2918   part,chapter,section,subsection,subsubsection,paragraph,%
2919   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2920   table,page,footnote,mpfootnote,mpfn}
2921 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2922   \bbl@ifunset{bbl@map@#1@language}%
2923   {\@nameuse{#1}}%
2924   {\@nameuse{bbl@map@#1@language}}}
2925 \def\bbl@inikv@labels#1#2{%
2926   \in@{.map}{#1}%
2927   \ifin@
2928     \ifx\bbl@KVP@labels\@nnil\else
2929       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
2930       \ifin@
2931         \def\bbl@tempc{#1}%
2932         \bbl@replace\bbl@tempc{.map}{}%
2933         \in@{, #2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
2934         \bbl@exp{%
2935           \gdef\<bbl@map@\bbl@tempc @language>%
2936           {\ifin@<#2>\else\\localecounter{#2}\fi}}%
2937         \bbl@foreach\bbl@list@the{%
2938           \bbl@ifunset{the##1}{%
2939             {\bbl@exp{\let\\bbl@tempd\<the##1>}}%
2940             \bbl@exp{%
2941               \\\bbl@sreplace\<the##1>%
2942               {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
2943               \\\bbl@sreplace\<the##1>%
2944               {\<\empty @\bbl@tempc>\<##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2945             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2946             \toks@\expandafter\expandafter\expandafter{%
2947               \csname the##1\endcsname}%
2948             \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%
2949           \fi}}%
2950         \fi
2951       \fi}

```

```

2952 %
2953 \else
2954 %
2955 % The following code is still under study. You can test it and make
2956 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
2957 % language dependent.
2958 \in@{enumerate.}{#1}%
2959 \ifin@
2960 \def\bb@tempa{#1}%
2961 \bb@replace\bb@tempa{enumerate.}{}%
2962 \def\bb@toreplace{#2}%
2963 \bb@replace\bb@toreplace{[ ]}{\nobreakspace{}}%
2964 \bb@replace\bb@toreplace{[ ]}{\csname the}%
2965 \bb@replace\bb@toreplace{[ ]}{\endcsname{}}%
2966 \toks@ \expandafter{\bb@toreplace}%
2967 % TODO. Execute only once:
2968 \bb@exp{%
2969 \\\bb@add\<extras\language>{%
2970 \\\babel@save\<labelenum\romannumeral\bb@tempa>%
2971 \def\<labelenum\romannumeral\bb@tempa>{\the\toks@}}%
2972 \\\bb@tglobal\<extras\language>}%
2973 \fi
2974 \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.

```

2975 \def\bb@chapttype{chapter}
2976 \ifx\@makechapterhead\undefined
2977 \let\bb@patchchapter\relax
2978 \else\ifx\thechapter\undefined
2979 \let\bb@patchchapter\relax
2980 \else\ifx\ps@headings\undefined
2981 \let\bb@patchchapter\relax
2982 \else
2983 \def\bb@patchchapter{%
2984 \global\let\bb@patchchapter\relax
2985 \gdef\bb@chfmt{%
2986 \bb@ifunset{bb@\bb@chapttype fmt@\language}%
2987 {\@chapapp\space\thechapter}%
2988 {\@nameuse{bb@\bb@chapttype fmt@\language}}}%
2989 \bb@add\appendix{\def\bb@chapttype{appendix}}% Not harmful, I hope
2990 \bb@replace\ps@headings{\@chapapp\ \thechapter}{\bb@chfmt}%
2991 \bb@replace\chaptermark{\@chapapp\ \thechapter}{\bb@chfmt}%
2992 \bb@replace\@makechapterhead{\@chapapp\space\thechapter}{\bb@chfmt}%
2993 \bb@tglobal\appendix
2994 \bb@tglobal\ps@headings
2995 \bb@tglobal\chaptermark
2996 \bb@tglobal\@makechapterhead}
2997 \let\bb@patchappendix\bb@patchchapter
2998 \fi\fi\fi
2999 \ifx\@part\undefined
3000 \let\bb@patchpart\relax
3001 \else
3002 \def\bb@patchpart{%
3003 \global\let\bb@patchpart\relax
3004 \gdef\bb@partformat{%
3005 \bb@ifunset{bb@partfmt@\language}%
3006 {\@partname\nobreakspace\thepart}%
3007 {\@nameuse{bb@partfmt@\language}}}%
3008 \bb@replace\@part{\@partname\nobreakspace\thepart}{\bb@partformat}%
3009 \bb@tglobal\@part}

```

3010 \fi

**Date.** Arguments (year, month, day) are *not* protected, on purpose. In \today, arguments are always gregorian, and therefore always converted with other calendars. TODO. Document

```
3011 \let\bbl@calendar\@empty
3012 \DeclareRobustCommand\localdate[1][\bbl@localdate{#1}]
3013 \def\bbl@localdate#1#2#3#4{%
3014   \begingroup
3015     \edef\bbl@they{#2}%
3016     \edef\bbl@them{#3}%
3017     \edef\bbl@thed{#4}%
3018     \edef\bbl@tempe{%
3019       \bbl@ifunset{bbl@calpr@\languagename}{\bbl@cl{calpr}},%
3020       #1}%
3021     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3022     \bbl@replace\bbl@tempe{ }{}%
3023     \bbl@replace\bbl@tempe{convert}{convert=%}%
3024     \let\bbl@ld@calendar\@empty
3025     \let\bbl@ld@variant\@empty
3026     \let\bbl@ld@convert\relax
3027     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld###1}{##2}}%
3028     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3029     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3030     \ifx\bbl@ld@calendar\@empty\else
3031       \ifx\bbl@ld@convert\relax\else
3032         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3033         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3034       \fi
3035     \fi
3036     \@nameuse{bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3037     \edef\bbl@calendar{% Used in \month..., too
3038       \bbl@ld@calendar
3039       \ifx\bbl@ld@variant\@empty\else
3040         .\bbl@ld@variant
3041       \fi}%
3042     \bbl@cased
3043     {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3044      \bbl@they\bbl@them\bbl@thed}%
3045   \endgroup}
3046 \def\bbl@printdate#1{%
3047   \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]]}
3048 \def\bbl@printdate@i#1[#2]#3#4#5{%
3049   \bbl@usedategrouptrue
3050   \@nameuse{bbl@ensure@#1}{\localdate[#2]{#3}{#4}{#5}}
3051 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3052 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3053   \bbl@trim@def\bbl@tempa{#1.#2}%
3054   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3055   {\bbl@trim@def\bbl@tempa{#3}%
3056    \bbl@trim\toks@{#5}%
3057    \@temptokena\expandafter{\bbl@savedate}%
3058    \bbl@exp{% Reverse order - in ini last wins
3059      \def\\bbl@savedate{%
3060        \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3061        \the\@temptokena}}}%
3062   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3063    {\lowercase{\def\bbl@tempb{#6}}%
3064     \bbl@trim@def\bbl@toreplace{#5}%
3065     \bbl@TG@@date
3066     \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3067     \ifx\bbl@savetoday\@empty
3068       \bbl@exp{% TODO. Move to a better place.
3069         \\AfterBabelCommands{%
```

```

3070         \gdef<\language name date>{\protect<\language name date >}%
3071         \gdef<\language name date >{\bbl@printdate{\language name}}}%
3072         \def\bbl@savetoday{%
3073             \SetString\today{%
3074                 <\language name date>[convert]%
3075                 {\the\year}{\the\month}{\the\day}}}%
3076         \fi}%
3077     }}

```

**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).

```

3078 \let\bbl@calendar\@empty
3079 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3080     \nameuse{bbl@ca#2}#1\@}
3081 \newcommand\BabelDateSpace{\nobreakspace}
3082 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3083 \newcommand\BabelDated[1]{\number#1}
3084 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3085 \newcommand\BabelDateM[1]{\number#1}
3086 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3087 \newcommand\BabelDateMMM[1]{%
3088     \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3089 \newcommand\BabelDatey[1]{\number#1}%
3090 \newcommand\BabelDateyy[1]{%
3091     \ifnum#1<10 0\number#1 %
3092     \else\ifnum#1<100 \number#1 %
3093     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3094     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3095     \else
3096         \bbl@error{limit-two-digits}{\number#1}%
3097     \fi\fi\fi\fi}
3098 \newcommand\BabelDateyyy[1]{\number#1} % TODO - add leading 0
3099 \newcommand\BabelDateU[1]{\number#1}%
3100 \def\bbl@replace@finish@iii#1{%
3101     \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3102 \def\bbl@TG@date{%
3103     \bbl@replace\bbl@toreplace[ ]{\BabelDateSpace}%
3104     \bbl@replace\bbl@toreplace[.]{\BabelDateDot}%
3105     \bbl@replace\bbl@toreplace[ d ]{\BabelDated{###3}}%
3106     \bbl@replace\bbl@toreplace[ dd ]{\BabelDatedd{###3}}%
3107     \bbl@replace\bbl@toreplace[ M ]{\BabelDateM{###2}}%
3108     \bbl@replace\bbl@toreplace[ MM ]{\BabelDateMM{###2}}%
3109     \bbl@replace\bbl@toreplace[ MMM ]{\BabelDateMMM{###2}}%
3110     \bbl@replace\bbl@toreplace[ y ]{\BabelDatey{###1}}%
3111     \bbl@replace\bbl@toreplace[ yy ]{\BabelDateyy{###1}}%
3112     \bbl@replace\bbl@toreplace[ yyyy ]{\BabelDateyyy{###1}}%
3113     \bbl@replace\bbl@toreplace[ U ]{\BabelDateU{###1}}%
3114     \bbl@replace\bbl@toreplace[ y ]{\bbl@datecncr[###1]}%
3115     \bbl@replace\bbl@toreplace[ U ]{\bbl@datecncr[###1]}%
3116     \bbl@replace\bbl@toreplace[ m ]{\bbl@datecncr[###2]}%
3117     \bbl@replace\bbl@toreplace[ d ]{\bbl@datecncr[###3]}%
3118     \bbl@replace@finish@iii\bbl@toreplace}
3119 \def\bbl@datecncr{\expandafter\bbl@xdatecncr\expandafter}
3120 \def\bbl@xdatecncr[#1#2]{\localenumeral{#2}{#1}}

```

## 4.21. French spacing (again)

For the following declarations, see issue #240. `\nonfrenchspacing` is set by document too early, so it's a hack.

```

3121 \AddToHook{begindocument/before}{%

```

```

3122 \let\bbl@normalsf\normalsfcodes
3123 \let\normalsfcodes\relax}
3124 \AtBeginDocument{%
3125 \ifx\bbl@normalsf\@empty
3126 \ifnum\sfcode`\.=\@m
3127 \let\normalsfcodes\frenchspacing
3128 \else
3129 \let\normalsfcodes\nonfrenchspacing
3130 \fi
3131 \else
3132 \let\normalsfcodes\bbl@normalsf
3133 \fi}

```

### Transforms.

```

3134 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3135 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3136 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3137 #1[#2]{#3}{#4}{#5}}
3138 \begingroup % A hack. TODO. Don't require a specific order
3139 \catcode`\%=12
3140 \catcode`\&=14
3141 \gdef\bbl@transforms#1#2#3{&%
3142 \directlua{
3143     local str = [==[#2]==]
3144     str = str:gsub('%.%d+%.%d+$', '')
3145     token.set_macro('babeltempa', str)
3146 }&%
3147 \def\babeltempc{&%
3148 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3149 \ifin@ \else
3150 \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&%
3151 \fi
3152 \ifin@
3153 \bbl@foreach\bbl@KVP@transforms{&%
3154 \bbl@xin@{: \babeltempa,}{,##1,}&%
3155 \ifin@ &% font:font:transform syntax
3156 \directlua{
3157     local t = {}
3158     for m in string.gmatch('##1'..' ':'(.)') do
3159         table.insert(t, m)
3160     end
3161     table.remove(t)
3162     token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3163 }&%
3164 \fi}&%
3165 \in@{.0$}{#2$}&%
3166 \ifin@
3167 \directlua{&% (\attribute) syntax
3168     local str = string.match([[ \bbl@KVP@transforms]],
3169         '%([[^%([-)%)][^%)]-\babeltempa')
3170     if str == nil then
3171         token.set_macro('babeltempb', '')
3172     else
3173         token.set_macro('babeltempb', ',attribute=' .. str)
3174     end
3175 }&%
3176 \toks@{#3}&%
3177 \bbl@exp{&%
3178 \\\g@addto@macro\\ \bbl@release@transforms{&%
3179 \relax &% Closes previous \bbl@transforms@aux
3180 \\\bbl@transforms@aux
3181 \\\#1{label=\babeltempa\babeltempb\babeltempc}&%
3182 { \language name }{ \the \toks@ } }&%

```



```

3183     \else
3184     \g@addto@macro\bb@release@transforms{, {#3}}&%
3185     \fi
3186     \fi}
3187 \endgroup

```

## 4.22. Handle language system

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

```

3188 \def\bb@provide@lsys#1{%
3189   \bb@i@funset{bb@lname@#1}%
3190   {\bb@load@info{#1}}%
3191   }%
3192   \bb@csarg\let{lsys@#1}\@empty
3193   \bb@i@funset{bb@sname@#1}{\bb@csarg\gdef{sname@#1}{Default}}{%
3194   \bb@i@funset{bb@sotf@#1}{\bb@csarg\gdef{sotf@#1}{DFLT}}{%
3195   \bb@csarg\bb@add@list{lsys@#1}{Script=\bb@cs{sname@#1}}%
3196   \bb@i@funset{bb@lname@#1}}%
3197   {\bb@csarg\bb@add@list{lsys@#1}{Language=\bb@cs{lname@#1}}%
3198   \ifcase\bb@engine\or\or
3199   \bb@i@funset{bb@prehc@#1}}%
3200   {\bb@exp{\\bb@ifblank{\bb@cs{prehc@#1}}%
3201   }%
3202   {\ifx\bb@xeno@hyph\undefined
3203     \global\let\bb@xeno@hyph\bb@xeno@hyph@
3204     \ifx\AtBeginDocument\@notprerr
3205       \expandafter\@secondoftwo % to execute right now
3206       \fi
3207     \AtBeginDocument{%
3208       \bb@patchfont{\bb@xeno@hyph}%
3209       {\expandafter\select@language\expandafter{\language}}%
3210     \fi}}%
3211   \fi
3212   \bb@csarg\bb@to@global{lsys@#1}}
3213 \def\bb@xeno@hyph@d{%
3214   \bb@ifset{bb@prehc@\language}%
3215   {\ifnum\hyphenchar\font=\defaulthyphenchar
3216     \iffontchar\font\bb@cl{prehc}\relax
3217     \hyphenchar\font\bb@cl{prehc}\relax
3218     \else\iffontchar\font"200B
3219       \hyphenchar\font"200B
3220     \else
3221       \bb@warning
3222       {Neither 0 nor ZERO WIDTH SPACE are available\\%
3223       in the current font, and therefore the hyphen\\%
3224       will be printed. Try changing the fontspec's\\%
3225       'HyphenChar' to another value, but be aware\\%
3226       this setting is not safe (see the manual).\\%
3227       Reported}%
3228       \hyphenchar\font\defaulthyphenchar
3229     \fi\fi
3230   \fi}%
3231   {\hyphenchar\font\defaulthyphenchar}}
3232   % \fi}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., 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).

```

3233 \def\bb@load@info#1{%
3234   \def\BabelBeforeIni##1##2{%
3235     \begingroup

```



```

3285 \def\bb@l@alphanumeric#1#2{%
3286 \expandafter\bb@l@alphanumeric@i\number#2 76543210\@@{#1}}
3287 \def\bb@l@alphanumeric@i#1#2#3#4#5#6#7#8\@@#9{%
3288 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3289 \bb@l@alphanumeric@ii{#9}00000#1\or
3290 \bb@l@alphanumeric@ii{#9}00000#1#2\or
3291 \bb@l@alphanumeric@ii{#9}00000#1#2#3\or
3292 \bb@l@alphanumeric@ii{#9}00000#1#2#3#4\else
3293 \bb@l@alphanum@invalid{>9999}%
3294 \fi}
3295 \def\bb@l@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3296 \bb@l@ifunset{bb@l@cntr@#1.F.\number#5#6#7#8@\languagename}%
3297 {\bb@l@cs{cntr@#1.4@\languagename}#5%
3298 \bb@l@cs{cntr@#1.3@\languagename}#6%
3299 \bb@l@cs{cntr@#1.2@\languagename}#7%
3300 \bb@l@cs{cntr@#1.1@\languagename}#8%
3301 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3302 \bb@l@ifunset{bb@l@cntr@#1.S.321@\languagename}{}%
3303 {\bb@l@cs{cntr@#1.S.321@\languagename}}%
3304 \fi}%
3305 {\bb@l@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}
3306 \def\bb@l@alphanum@invalid#1{%
3307 \bb@l@error{alphabetic-too-large}{#1}{}}

```

## 4.24. Casing

```

3308 \newcommand\BabelUppercaseMapping[3]{%
3309 \DeclareUppercaseMapping[\@nameuse{bb@casing@#1}]{#2}{#3}}
3310 \newcommand\BabelTitlecaseMapping[3]{%
3311 \DeclareTitlecaseMapping[\@nameuse{bb@casing@#1}]{#2}{#3}}
3312 \newcommand\BabelLowercaseMapping[3]{%
3313 \DeclareLowercaseMapping[\@nameuse{bb@casing@#1}]{#2}{#3}}

The parser for casing and casing. (variant).
3314 \ifcase\bb@engine % Converts utf8 to its code (expandable)
3315 \def\bb@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3316 \else
3317 \def\bb@uftocode#1{\expandafter`\string#1}
3318 \fi
3319 \def\bb@casemapping#1#2#3{% 1:variant
3320 \def\bb@tempa##1 ##2{% Loop
3321 \bb@casemapping@i{##1}%
3322 \ifx\@empty##2\else\bb@afterfi\bb@tempa##2\fi}%
3323 \edef\bb@templ{\@nameuse{bb@casing@#2}#1}% Language code
3324 \def\bb@tempe{0}% Mode (upper/lower...)
3325 \def\bb@tempc{#3}% Casing list
3326 \expandafter\bb@tempa\bb@tempc\@empty}
3327 \def\bb@casemapping@i#1{%
3328 \def\bb@tempb{#1}%
3329 \ifcase\bb@engine % Handle utf8 in pdftex, by surrounding chars with {}
3330 \@nameuse{regex_replace_all:nnN}%
3331 {[{\x{c0}-\x{ff}][\x{80}-\x{bf}]}*}{\0}}\bb@tempb
3332 \else
3333 \@nameuse{regex_replace_all:nnN}{.}{\0}}\bb@tempb % TODO. needed?
3334 \fi
3335 \expandafter\bb@casemapping@ii\bb@tempb\@@}
3336 \def\bb@casemapping@ii#1#2#3\@@{%
3337 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3338 \ifin@
3339 \edef\bb@tempe{%
3340 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3341 \else
3342 \ifcase\bb@tempe\relax
3343 \DeclareUppercaseMapping[\bb@templ]{\bb@uftocode{#1}]{#2}%

```

```

3344     \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3345     \or
3346     \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3347     \or
3348     \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3349     \or
3350     \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3351     \fi
3352 \fi}

```

## 4.25. Getting info

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

3353 \def\bbl@localeinfo#1#2{%
3354   \bbl@ifunset{bbl@info@#2}{#1}%
3355   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3356    {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}%
3357 \newcommand\localeinfo[1]{%
3358   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3359   \bbl@afterelse\bbl@localeinfo}%
3360 \else
3361   \bbl@localeinfo
3362   {\bbl@error{no-ini-info}{}}}%
3363   {#1}%
3364 \fi}
3365 % \@namedef{bbl@info@name.locale}{lcname}
3366 \@namedef{bbl@info@tag.ini}{lini}
3367 \@namedef{bbl@info@name.english}{elname}
3368 \@namedef{bbl@info@name.opentype}{lname}
3369 \@namedef{bbl@info@tag.bcp47}{tbcpl}
3370 \@namedef{bbl@info@language.tag.bcp47}{lbcpl}
3371 \@namedef{bbl@info@tag.opentype}{lotf}
3372 \@namedef{bbl@info@script.name}{esname}
3373 \@namedef{bbl@info@script.name.opentype}{sname}
3374 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3375 \@namedef{bbl@info@script.tag.opentype}{soft}
3376 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3377 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3378 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3379 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3380 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}

```

With version 3.75 `\BabelEnsureInfo` is executed always, but there is an option to disable it.

```

3381 <<{*More package options}>> ≡
3382 \DeclareOption{ensureinfo=off}{}
3383 <</More package options>>
3384 \let\bbl@ensureinfo\@gobble
3385 \newcommand\BabelEnsureInfo{%
3386   \ifx\InputIfFileExists\@undefined\else
3387   \def\bbl@ensureinfo##1{%
3388     \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
3389   \fi
3390   \bbl@foreach\bbl@loaded{%
3391     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3392     \def\languagename{##1}%
3393     \bbl@ensureinfo{##1}}}%
3394 \@ifpackagewith{babel}{ensureinfo=off}{}%
3395 {\AtEndOfPackage{% Test for plain.
3396   \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

More general, but non-expandable, is `\getLocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

3397 \newcommand\getlocaleproperty{%
3398 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3399 \def\bbl@getproperty@s#1#2#3{%
3400 \let#1\relax
3401 \def\bbl@elt##1##2##3{%
3402 \bbl@ifsamestring{##1/##2}{#3}%
3403 {\providecommand#1{##3}%
3404 \def\bbl@elt####1####2####3{}}%
3405 {}}%
3406 \bbl@cs{inidata@#2}}%
3407 \def\bbl@getproperty@x#1#2#3{%
3408 \bbl@getproperty@s{#1}{#2}{#3}%
3409 \ifx#1\relax
3410 \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3411 \fi}
3412 \let\bbl@ini@loaded\@empty
3413 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3414 \def\ShowLocaleProperties#1{%
3415 \typeout{}}%
3416 \typeout{*** Properties for language '#1' ***}
3417 \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3418 \@nameuse{\bbl@inidata@#1}%
3419 \typeout{*****}}

```

#### 4.26. BCP 47 related commands

```

3420 \newif\ifbbl@bcppallowed
3421 \bbl@bcppallowedfalse
3422 \def\bbl@autoload@options{import}
3423 \def\bbl@provide@locale{%
3424 \ifx\babelprovide\undefined
3425 \bbl@error{base-on-the-fly}{}{}%
3426 \fi
3427 \let\bbl@auxname\language\language % Still necessary. %^^A TODO
3428 \bbl@ifunset{\bbl@bcp@map@\language}{}% Move uplevel??
3429 {\edef\language{\@nameuse{\bbl@bcp@map@\language}}}%
3430 \ifbbl@bcppallowed
3431 \expandafter\ifx\csname date\language\endcsname\relax
3432 \expandafter
3433 \bbl@bcplookup\language-\@empty-\@empty-\@empty@@
3434 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3435 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3436 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
3437 \expandafter\ifx\csname date\language\endcsname\relax
3438 \let\bbl@initoload\bbl@bcp
3439 \bbl@exp{\babelprovide[\bbl@autoload@bcppoptions]{\language}}%
3440 \let\bbl@initoload\relax
3441 \fi
3442 \bbl@csarg\xdef{\bbl@bcp}{\localename}%
3443 \fi
3444 \fi
3445 \fi
3446 \expandafter\ifx\csname date\language\endcsname\relax
3447 \IfFileExists{\babel-\language.tex}%
3448 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3449 {}%
3450 \fi}

```

$\LaTeX$  needs to know the BCP 47 codes for some features. For that, it expects `\BCPdata` to be defined. While language, region, script, and variant are recognized, extension. `(s)` for singletons may change.

Still somewhat hackish. WIP. Note `\str_if_eq:nnTF` is fully expandable (`\bbl@ifsamestring` isn't). The argument is the prefix to `tag.bcp47`.

```

3451 \providecommand\BCPdata{}

```

```

3452 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3453 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1@empty\@empty\@empty}
3454 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3455 \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3456 {\bbl@bcpdata@ii#6}\bbl@main@language}%
3457 {\bbl@bcpdata@ii#1#2#3#4#5#6}\languagename}}%
3458 \def\bbl@bcpdata@ii#1#2{%
3459 \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3460 {\bbl@error{unknown-ini-field}{#1}{}}}%
3461 {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3462 {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}}
3463 \fi
3464 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3465 \@namedef{bbl@info@tag.tag.bcp47}{tbcpl} % For \BCPdata

```

## 5. Adjusting the Babel behavior

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

```

3466 \newcommand\babeladjust[1]{% TODO. Error handling.
3467 \bbl@forkv{#1}{%
3468 \bbl@ifunset{bbl@ADJ@##1@##2}%
3469 {\bbl@cs{ADJ@##1}{##2}}%
3470 {\bbl@cs{ADJ@##1@##2}}}
3471 %
3472 \def\bbl@adjust@lua#1#2{%
3473 \ifvmode
3474 \ifnum\currentgrouplevel=\z@
3475 \directlua{ Babel.#2 }%
3476 \expandafter\expandafter\expandafter\@gobble
3477 \fi
3478 \fi
3479 {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3480 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3481 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3482 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3483 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3484 \@namedef{bbl@ADJ@bidi.text@on}{%
3485 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3486 \@namedef{bbl@ADJ@bidi.text@off}{%
3487 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3488 \@namedef{bbl@ADJ@bidi.math@on}{%
3489 \let\bbl@noamsmath\@empty}
3490 \@namedef{bbl@ADJ@bidi.math@off}{%
3491 \let\bbl@noamsmath\relax}
3492 %
3493 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3494 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3495 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3496 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3497 %
3498 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3499 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3500 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3501 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3502 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3503 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3504 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3505 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3506 \@namedef{bbl@ADJ@justify.arabic@on}{%
3507 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3508 \@namedef{bbl@ADJ@justify.arabic@off}{%
3509 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3510 %

```

```

3511 \def\bb@adjust@layout#1{%
3512   \ifvmode
3513     #1%
3514   \expandafter\@gobble
3515   \fi
3516   {\bb@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3517 \@namedef{bb@ADJ@layout.tabular@on}{%
3518   \ifnum\bb@tabular@mode=\tw@
3519     \bb@adjust@layout{\let\@tabular\bb@NL@@@tabular}%
3520   \else
3521     \chardef\bb@tabular@mode\@ne
3522   \fi}
3523 \@namedef{bb@ADJ@layout.tabular@off}{%
3524   \ifnum\bb@tabular@mode=\tw@
3525     \bb@adjust@layout{\let\@tabular\bb@OL@@@tabular}%
3526   \else
3527     \chardef\bb@tabular@mode\@z@
3528   \fi}
3529 \@namedef{bb@ADJ@layout.lists@on}{%
3530   \bb@adjust@layout{\let\list\bb@NL@list}}
3531 \@namedef{bb@ADJ@layout.lists@off}{%
3532   \bb@adjust@layout{\let\list\bb@OL@list}}
3533 %
3534 \@namedef{bb@ADJ@autoload.bcp47@on}{%
3535   \bb@bcppallowedtrue}
3536 \@namedef{bb@ADJ@autoload.bcp47@off}{%
3537   \bb@bcppallowedfalse}
3538 \@namedef{bb@ADJ@autoload.bcp47.prefix}#1{%
3539   \def\bb@bcp@prefix{#1}}
3540 \def\bb@bcp@prefix{bcp47-}
3541 \@namedef{bb@ADJ@autoload.options}#1{%
3542   \def\bb@autoload@options{#1}}
3543 \def\bb@autoload@bcptoptions{import}
3544 \@namedef{bb@ADJ@autoload.bcp47.options}#1{%
3545   \def\bb@autoload@bcptoptions{#1}}
3546 \newif\ifbb@bcptname
3547 \@namedef{bb@ADJ@bcp47.toname@on}{%
3548   \bb@bcptnametrue}
3549 \BabelEnsureInfo
3550 \@namedef{bb@ADJ@bcp47.toname@off}{%
3551   \bb@bcptnamefalse}
3552 \@namedef{bb@ADJ@prehyphenation.disable@nohyphenation}{%
3553   \directlua{ Babel.ignore_pre_char = function(node)
3554     return (node.lang == \the\csname \@nohyphenation\endcsname)
3555   end }}
3556 \@namedef{bb@ADJ@prehyphenation.disable@off}{%
3557   \directlua{ Babel.ignore_pre_char = function(node)
3558     return false
3559   end }}
3560 \@namedef{bb@ADJ@interchar.disable@nohyphenation}{%
3561   \def\bb@ignoreinterchar{%
3562     \ifnum\language=\@nohyphenation
3563       \expandafter\@gobble
3564     \else
3565       \expandafter\@firstofone
3566     \fi}}
3567 \@namedef{bb@ADJ@interchar.disable@off}{%
3568   \let\bb@ignoreinterchar\@firstofone}
3569 \@namedef{bb@ADJ@select.write@shift}{%
3570   \let\bb@restorelastskip\relax
3571   \def\bb@savelastskip{%
3572     \let\bb@restorelastskip\relax
3573   \ifvmode

```

```

3574     \ifdim\lastskip=\z@
3575     \let\bbl@restorelastskip\nobreak
3576     \else
3577     \bbl@exp{%
3578     \def\\bbl@restorelastskip{%
3579     \skip@=\the\lastskip
3580     \\nobreak \vskip-\skip@ \vskip\skip@}}%
3581     \fi
3582     \fi}}
3583 \@namedef{bbl@ADJ@select.write@keep}{%
3584 \let\bbl@restorelastskip\relax
3585 \let\bbl@savelastskip\relax}
3586 \@namedef{bbl@ADJ@select.write@omit}{%
3587 \AddBabelHook{babel-select}{beforestart}{%
3588 \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3589 \let\bbl@restorelastskip\relax
3590 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3591 \@namedef{bbl@ADJ@select.encoding@off}{%
3592 \let\bbl@encoding@select@off\@empty}

```

## 5.1. Cross referencing macros

The  $\LaTeX$  book states:

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

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

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

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

```

3593 << *More package options >> ≡
3594 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3595 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3596 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3597 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3598 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3599 << /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.

```

3600 \bbl@trace{Cross referencing macros}
3601 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3602 \def\@newl@bel#1#2#3{%
3603 {\@safe@activestrue
3604 \bbl@ifunset{#1@#2}%
3605 \relax
3606 {\gdef\@multiplelabels{%
3607 \@latex@warning@no@line{There were multiply-defined labels}}%
3608 \@latex@warning@no@line{Label `#2' multiply defined}}%
3609 \global\@namedef{#1@#2}{#3}}

```

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

```

3610 \CheckCommand*\@testdef[3]{%
3611 \def\reserved@a{#3}%
3612 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3613 \else
3614 \@tempwattrue
3615 \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 `\@newlabel` 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.

```

3616 \def\@testdef#1#2#3{% TODO. With @samestring?
3617   \@safe@activestru
3618   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3619   \def\bbl@tempb{#3}%
3620   \@safe@activestru
3621   \ifx\bbl@tempa\relax
3622   \else
3623     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3624   \fi
3625   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3626   \ifx\bbl@tempa\bbl@tempb
3627   \else
3628     \@tempswatru
3629   \fi}
3630 \fi

```

### **\ref**

**\pageref** 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.

```

3631 \bbl@xin@{R}\bbl@opt@safe
3632 \ifin@
3633 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3634 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3635   {\expandafter\strip@prefix\meaning\ref}%
3636 \ifin@
3637   \bbl@redefine\@kernel@ref#1{%
3638     \@safe@activestru\org@@kernel@ref{#1}\@safe@activestru}
3639   \bbl@redefine\@kernel@pageref#1{%
3640     \@safe@activestru\org@@kernel@pageref{#1}\@safe@activestru}
3641   \bbl@redefine\@kernel@sref#1{%
3642     \@safe@activestru\org@@kernel@sref{#1}\@safe@activestru}
3643   \bbl@redefine\@kernel@spageref#1{%
3644     \@safe@activestru\org@@kernel@spageref{#1}\@safe@activestru}
3645   \else
3646     \bbl@redefinero\ref#1{%
3647       \@safe@activestru\org@ref{#1}\@safe@activestru}
3648     \bbl@redefinero\pageref#1{%
3649       \@safe@activestru\org@pageref{#1}\@safe@activestru}
3650   \fi
3651 \else
3652   \let\org@ref\ref
3653   \let\org@pageref\pageref
3654 \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.

```

3655 \bbl@xin@{B}\bbl@opt@safe
3656 \ifin@
3657   \bbl@redefine\@citex[#1]#2{%
3658     \@safe@activestru\edef\bbl@tempa{#2}\@safe@activestru}
3659   \org@@citex[#1]{\bbl@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.

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.)

```
3660 \AtBeginDocument{%
3661   \@ifpackageloaded{natbib}{%
3662     \def\@citex[#1][#2]#3{%
3663       \@safe@activestrue\edef\bbl@tempa{#3}\@safe@activestrue
3664       \org@citex[#1][#2]{\bbl@tempa}}%
3665     }{}}
```

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

```
3666 \AtBeginDocument{%
3667   \@ifpackageloaded{cite}{%
3668     \def\@citex[#1]#2{%
3669       \@safe@activestrue\org@citex[#1][#2]\@safe@activestrue}%
3670     }{}}
```

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

```
3671 \bbl@redefine\nocite#1{%
3672   \@safe@activestrue\org@nocite{#1}\@safe@activestrue}
```

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

```
3673 \bbl@redefine\bibcite{%
3674   \bbl@cite@choice
3675   \bibcite}
```

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

```
3676 \def\bbl@bibcite#1#2{%
3677   \org@bibcite{#1}{\@safe@activestrue#2}}
```

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

```
3678 \def\bbl@cite@choice{%
3679   \global\let\bibcite\bbl@bibcite
3680   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3681     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3682     \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.

```
3683 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3684 \bbl@redefine\@bibitem#1{%
3685   \@safe@activestrue\org@bibitem{#1}\@safe@activestrue}
3686 \else
3687   \let\org@nocite\nocite
3688   \let\org@citex\@citex
```

```

3689 \let\org@bibcite\bibcite
3690 \let\org@@bibitem@\bibitem
3691 \fi

```

## 5.2. Layout

```

3692 \newcommand\BabelPatchSection[1]{%
3693 \@ifundefined{#1}{%
3694 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3695 \@namedef{#1}{%
3696 \ifstar{\bbl@presec@s{#1}}%
3697 {\@dblarg{\bbl@presec@x{#1}}}}
3698 \def\bbl@presec@x#1[#2]#3{%
3699 \bbl@exp{%
3700 \\\select@language@x{\bbl@main@language}%
3701 \\\bbl@cs{sspre@#1}%
3702 \\\bbl@cs{ss@#1}%
3703 [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3704 {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3705 \\\select@language@x{\language}}
3706 \def\bbl@presec@s#1#2{%
3707 \bbl@exp{%
3708 \\\select@language@x{\bbl@main@language}%
3709 \\\bbl@cs{sspre@#1}%
3710 \\\bbl@cs{ss@#1}*%
3711 {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3712 \\\select@language@x{\language}}
3713 \IfBabelLayout{sectioning}%
3714 {\BabelPatchSection{part}%
3715 \BabelPatchSection{chapter}%
3716 \BabelPatchSection{section}%
3717 \BabelPatchSection{subsection}%
3718 \BabelPatchSection{subsubsection}%
3719 \BabelPatchSection{paragraph}%
3720 \BabelPatchSection{subparagraph}%
3721 \def\babel@toc#1{%
3722 \select@language@x{\bbl@main@language}}}{
3723 \IfBabelLayout{captions}%
3724 {\BabelPatchSection{caption}}{

```

## 5.3. 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.

```

3725 \bbl@trace{Marks}
3726 \IfBabelLayout{sectioning}
3727 {\ifx\bbl@opt@headfoot@nnil
3728 \g@addto@macro\@resetactivechars{%
3729 \set@typeset@protect
3730 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3731 \let\protect\noexpand
3732 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3733 \edef\thepage{%
3734 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3735 \fi}%
3736 \fi}
3737 {\ifbbl@single\else
3738 \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3739 \markright#1}%

```

```

3740 \bbl@ifblank{#1}%
3741 {\org@markright{}}%
3742 {\toks@{#1}}%
3743 \bbl@exp{%
3744   \\org@markright{\\protect\\foreignlanguage{\\language}%
3745   {\\protect\\bbl@restore@actives\the\toks@}}}%

```

## **\markboth**

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

```

3746 \ifx\@mkboth\markboth
3747 \def\bbl@tempc{\let\@mkboth\markboth}%
3748 \else
3749 \def\bbl@tempc{%
3750 \fi
3751 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinero bust
3752 \markboth#1#2{%
3753 \protected@edef\bbl@tempb##1{%
3754 \protect\foreignlanguage
3755 {\\language}{\\protect\bbl@restore@actives##1}}%
3756 \bbl@ifblank{#1}%
3757 {\toks@{}}%
3758 {\toks@\expandafter{\bbl@tempb{#1}}}%
3759 \bbl@ifblank{#2}%
3760 {\@temptokena{}}%
3761 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3762 \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}%
3763 \bbl@tempc
3764 \fi} % end ifbbl@single, end \IfBabelLayout

```

## 5.4. Other packages

### 5.4.1. `ifthen`

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

```

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

```

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

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.

```

3765 \bbl@trace{Preventing clashes with other packages}
3766 \ifx\org@ref\undefined\else
3767 \bbl@xin@{R}\bbl@opt@safe
3768 \ifin@
3769 \AtBeginDocument{%
3770 \@ifpackageloaded{ifthen}{%
3771 \bbl@redefine@long\ifthenelse#1#2#3{%
3772 \let\bbl@temp@pref\pageref

```

```

3773     \let\pageref\org@pageref
3774     \let\bbl@temp@ref\ref
3775     \let\ref\org@ref
3776     \@safe@activestru
3777     \org@ifthenelse{#1}%
3778         {\let\pageref\bbl@temp@pref
3779          \let\ref\bbl@temp@ref
3780          \@safe@activesfalse
3781          #2}%
3782     {\let\pageref\bbl@temp@pref
3783     \let\ref\bbl@temp@ref
3784     \@safe@activesfalse
3785     #3}%
3786     }%
3787     }{}%
3788     }
3789 \fi

```

#### 5.4.2. varioref

**\@@vpageref**

**\vrefpagemum**

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

```

3790 \AtBeginDocument{%
3791   \ifpackageloaded{varioref}{%
3792     \bbl@redefine\@@vpageref#1[#2]#3{%
3793       \@safe@activestru
3794       \org@@@vpageref{#1}[#2]{#3}%
3795       \@safe@activesfalse}%
3796     \bbl@redefine\vrefpagemum#1#2{%
3797       \@safe@activestru
3798       \org@vrefpagemum{#1}#2}%
3799     \@safe@activesfalse}%

```

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

```

3800   \expandafter\def\csname Ref \endcsname#1{%
3801     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3802   }{}%
3803 }
3804 \fi

```

#### 5.4.3. hhlne

**\hhlne** Delaying the activation of the shorthand characters has introduced a problem with the `hhlne` 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.

```

3805 \AtEndOfPackage{%
3806   \AtBeginDocument{%
3807     \ifpackageloaded{hhlne}%
3808       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3809        \else
3810          \makeatletter
3811          \def\@currname{hhlne}\input{hhlne.sty}\makeatother
3812          \fi}%
3813       {}}}

```

**\substitutefontfamily** *Deprecated.* It creates an fd file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names. Use the tools provided by  $\TeX$  (`\DeclareFontFamilySubstitution`).

```

3814 \def\substitutefontfamily#1#2#3{%
3815   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3816   \immediate\write15{%
3817     \string\ProvidesFile{#1#2.fd}%
3818     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3819     \space generated font description file]^J
3820     \string\DeclareFontFamily{#1}{#2}{ }^^J
3821     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^^J
3822     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^^J
3823     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^^J
3824     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^^J
3825     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^^J
3826     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^^J
3827     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^^J
3828     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^^J
3829   }%
3830   \closeout15
3831 }
3832 \@onlypreamble\substitutefontfamily

```

## 5.5. Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\TeX$  and  $\LaTeX$  always come out in the right encoding. There is a list of non-ASCII encodings. 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**

```

3833 \bbl@trace{Encoding and fonts}
3834 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3835 \newcommand\BabelNonText{TS1,T3,TS3}
3836 \let\org@TeX\TeX
3837 \let\org@LaTeX\LaTeX
3838 \let\ensureascii@firstofone
3839 \let\asciientoding@empty
3840 \AtBeginDocument{%
3841   \def\@elt#1{,#1,}%
3842   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3843   \let\@elt\relax
3844   \let\bbl@tempb@empty
3845   \def\bbl@tempc{OT1}%
3846   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3847     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3848   \bbl@foreach\bbl@tempa{%
3849     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3850     \ifin@
3851       \def\bbl@tempb{#1}% Store last non-ascii
3852     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3853       \ifin@else
3854         \def\bbl@tempc{#1}% Store last ascii
3855         \fi
3856       \fi}%
3857   \ifx\bbl@tempb@empty\else
3858     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3859     \ifin@else
3860       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3861     \fi
3862     \let\asciientoding\bbl@tempc

```

```

3863 \renewcommand\ensureascii[1]{%
3864   {\fontencoding{\asciencoding}\selectfont#1}}%
3865 \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3866 \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3867 \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.

```

3868 \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.

```

3869 \AtBeginDocument{%
3870 \@ifpackageloaded{fontspec}%
3871   {\xdef\latinencoding{%
3872     \ifx\UTFencname\@undefined
3873       EU\ifcase\bbl@engine\or2\or1\fi
3874     \else
3875       \UTFencname
3876     \fi}}%
3877 \gdef\latinencoding{OT1}%
3878 \ifx\cf@encoding\bbl@t@one
3879   \xdef\latinencoding{\bbl@t@one}%
3880 \else
3881   \def\@elt#1{,#1,}%
3882   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3883   \let\@elt\relax
3884   \bbl@xin@{,T1,}\bbl@tempa
3885   \ifin@
3886     \xdef\latinencoding{\bbl@t@one}%
3887   \fi
3888 \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.

```

3889 \DeclareRobustCommand{\latintext}{%
3890 \fontencoding{\latinencoding}\selectfont
3891 \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.

```

3892 \ifx\@undefined\DeclareTextFontCommand
3893 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3894 \else
3895 \DeclareTextFontCommand{\textlatin}{\latintext}
3896 \fi

```

For several functions, we need to execute some code with `\selectfont`. With  $\TeX$  2021-06-01, there is a hook for this purpose.

```

3897 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

## 5.6. Basic bidi support

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

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

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

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

```

3898 \bbl@trace{Loading basic (internal) bidi support}
3899 \ifodd\bbl@engine
3900 \else % TODO. Move to txtbabel. Any xe+lua bidi
3901 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3902 \bbl@error{bidi-only-lua}{\}\}\}\}%
3903 \let\bbl@beforeforeign\leavevmode
3904 \AtEndOfPackage{%
3905 \EnableBabelHook{babel-bidi}%
3906 \bbl@xebidipar}
3907 \fi\fi
3908 \def\bbl@loadxebidi#1{%
3909 \ifx\RTLfootnotetext\@undefined
3910 \AtEndOfPackage{%
3911 \EnableBabelHook{babel-bidi}%
3912 \ifx\fontspec\@undefined
3913 \usepackage{fontspec}% bidi needs fontspec
3914 \fi
3915 \usepackage#1{bidi}%
3916 \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3917 \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3918 \ifnum\@nameuse{\bbl@wdir@\languagename}=\tw@ % 'AL' bidi
3919 \bbl@digitsdotdash % So ignore in 'R' bidi
3920 \fi}}%
3921 \fi}
3922 \ifnum\bbl@bidimode>200 % Any xe bidi=
3923 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3924 \bbl@tentative{bidi=bidi}
3925 \bbl@loadxebidi{}
3926 \or
3927 \bbl@loadxebidi{[rldocument]}
3928 \or
3929 \bbl@loadxebidi{}
3930 \fi
3931 \fi
3932 \fi
3933 % TODO? Separate:
3934 \ifnum\bbl@bidimode=\@ne % bidi=default
3935 \let\bbl@beforeforeign\leavevmode
3936 \ifodd\bbl@engine % lua
3937 \newattribute\bbl@attr@dir
3938 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3939 \bbl@exp{\output{\bodydir\pagedir\the\output}}
3940 \fi
3941 \AtEndOfPackage{%
3942 \EnableBabelHook{babel-bidi}% pdf/luaxe

```



```

3943 \ifodd\bbl@engine\else % pdf/xe
3944 \bbl@xebidipar
3945 \fi}
3946 \fi

```

Now come the macros used to set the direction when a language is switched. Testing are based on script names, because it's the user interface (including language and script in \babelprovide. First the (mostly) common macros.

```

3947 \bbl@trace{Macros to switch the text direction}
3948 \def\bbl@alscripts{%
3949   ,Arabic,Syriac,Thaana,Hanifi_Rohingya,Hanifi,Sogdian,}
3950 \def\bbl@rscripts{%
3951   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
3952   Hatran,Hebrew,Imperial_Aramaic,Inscriptional_Pahlavi,%
3953   Inscriptional_Parthian,Kharoshthi,Lydian,Mandaic,Manichaean,%
3954   Mende_Kikakui,Meroitic_Cursive,Meroitic_Hieroglyphs,Nabataean,%
3955   Nko,Old_Hungarian,Old_North_Arabian,Old_Sogdian,%
3956   Old_South_Arabian,Old_Turkic,Old_Uyghur,Palmyrene,Phoenician,%
3957   Psalter_Pahlavi,Samaritan,Yezidi,Mandaean,%
3958   Meroitic,N'Ko,Orkhon,Todhri}
3959 \def\bbl@provide@dirs#1{%
3960   \bbl@xin@{\csname bbl@sname@#1\endcsname}\bbl@alscripts\bbl@rscripts}%
3961   \ifin@
3962     \global\bbl@csarg\chardef{wdir@#1}\@ne
3963     \bbl@xin@{\csname bbl@sname@#1\endcsname}\bbl@alscripts}%
3964     \ifin@
3965     \global\bbl@csarg\chardef{wdir@#1}\tw@
3966     \fi
3967   \else
3968     \global\bbl@csarg\chardef{wdir@#1}\z@
3969   \fi
3970 \ifodd\bbl@engine
3971   \bbl@csarg\ifcase{wdir@#1}%
3972     \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3973   \or
3974     \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3975   \or
3976     \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3977   \fi
3978 \fi}
3979 \def\bbl@switchdir{%
3980   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%}
3981   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%}
3982   \bbl@exp{\\bbl@setdirs\bbl@c{wdir}}
3983 \def\bbl@setdirs#1{% TODO - math
3984   \ifcase\bbl@select@type % TODO - strictly, not the right test
3985     \bbl@bodydir{#1}%
3986     \bbl@pardir{#1}% <- Must precede \bbl@textdir
3987   \fi
3988   \bbl@textdir{#1}}
3989 \ifnum\bbl@bidimode>\z@
3990   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3991   \DisableBabelHook{babel-bidi}
3992 \fi

```

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

```

3993 \ifodd\bbl@engine % luatex=1
3994 \else % pdftex=0, xetex=2
3995   \newcount\bbl@dirlevel
3996   \chardef\bbl@thetextdir\z@
3997   \chardef\bbl@thepardir\z@
3998   \def\bbl@textdir#1{%
3999     \ifcase#1\relax
4000     \chardef\bbl@thetextdir\z@

```

```

4001     \nameuse{setlatin}%
4002     \bbl@textdir@i\beginL\endL
4003   \else
4004     \chardef\bbl@thetextdir@ne
4005     \nameuse{setnonlatin}%
4006     \bbl@textdir@i\beginR\endR
4007   \fi}
4008 \def\bbl@textdir@i#1#2{%
4009   \ifhmode
4010     \ifnum\currentgrouplevel>\z@
4011       \ifnum\currentgrouplevel=\bbl@dirlevel
4012         \bbl@error{multiple-bidi}{\}\}%
4013         \bgroup\aftergroup#2\aftergroup\egroup
4014       \else
4015         \ifcase\currentgrouptype\or % 0 bottom
4016           \aftergroup#2% 1 simple {}
4017         \or
4018           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4019         \or
4020           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4021         \or\or\or % vbox vtop align
4022         \or
4023           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4024         \or\or\or\or\or\or % output math disc insert vcent mathchoice
4025         \or
4026           \aftergroup#2% 14 \begingroup
4027         \else
4028           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4029       \fi
4030     \fi
4031     \bbl@dirlevel\currentgrouplevel
4032   \fi
4033   #1%
4034 \fi}
4035 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4036 \let\bbl@bodydir@gobble
4037 \let\bbl@pagedir@gobble
4038 \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).

```

4039 \def\bbl@xebidipar{%
4040   \let\bbl@xebidipar\relax
4041   \TeXeTstate\@ne
4042   \def\bbl@xeeverypar{%
4043     \ifcase\bbl@thepardir
4044       \ifcase\bbl@thetextdir\else\beginR\fi
4045     \else
4046       {\setbox\z@\lastbox\beginR\box\z@}%
4047     \fi}%
4048   \AddToHook{para/begin}{\bbl@xeeverypar}}
4049 \ifnum\bbl@bidimode>200 % Any xe bidi=
4050   \let\bbl@textdir@i@gobbletwo
4051   \let\bbl@xebidipar@empty
4052   \AddBabelHook{bidi}{foreign}{%
4053     \ifcase\bbl@thetextdir
4054       \BabelWrapText{\LR{##1}}%
4055     \else
4056       \BabelWrapText{\RL{##1}}%
4057     \fi}
4058   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4059 \fi

```

```

4060 \fi
    A tool for weak L (mainly digits). We also disable warnings with hyperref.
4061 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4062 \AtBeginDocument{%
4063   \ifx\pdfstringdefDisableCommands\undefined\else
4064     \ifx\pdfstringdefDisableCommands\relax\else
4065       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4066     \fi
4067   \fi}

```

## 5.7. Local Language Configuration

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

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

```

4068 \bbl@trace{Local Language Configuration}
4069 \ifx\loadlocalcfg\undefined
4070   \@ifpackagewith{babel}{noconfigs}%
4071     {\let\loadlocalcfg\@gobble}%
4072   {\def\loadlocalcfg#1{%
4073     \InputIfFileExists{#1.cfg}%
4074     {\typeout{*****^J%
4075               * Local config file #1.cfg used^^J%
4076               *}}%
4077     \@empty}}
4078 \fi

```

## 5.8. 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).

```

4079 \bbl@trace{Language options}
4080 \let\bbl@afterlang\relax
4081 \let\BabelModifiers\relax
4082 \let\bbl@loaded\@empty
4083 \def\bbl@load@language#1{%
4084   \InputIfFileExists{#1.ldf}%
4085   {\edef\bbl@loaded{\CurrentOption
4086     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4087     \expandafter\let\expandafter\bbl@afterlang
4088     \csname\CurrentOption.ldf-h@k\endcsname
4089     \expandafter\let\expandafter\BabelModifiers
4090     \csname bbl@mod@\CurrentOption\endcsname
4091     \bbl@exp{\AtBeginDocument{%
4092       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4093     {\IfFileExists{babel-#1.tex}%
4094       {\def\bbl@tempa{%
4095         .\There is a locale ini file for this language.\%
4096         If it's the main language, try adding `provide=*'\%
4097         to the babel package options}}%
4098       {\let\bbl@tempa\empty}%
4099       \bbl@error{unknown-package-option}{}}}}

```

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.

```

4100 \def\bbl@try@load@lang#1#2#3{%

```

```

4101 \IfFileExists{\CurrentOption.ldf}%
4102   {\bbl@load@language{\CurrentOption}}%
4103   {#1\bbl@load@language{#2}#3}}
4104 %
4105 \DeclareOption{friulian}{\bbl@try@load@lang{}{friulan}{}}
4106 \DeclareOption{hebrew}{%
4107   \ifcase\bbl@engine\or
4108     \bbl@error{only-pdftex-lang}{hebrew}{\luatex}{}}%
4109   \fi
4110   \input{rlbabel.def}%
4111   \bbl@load@language{hebrew}}
4112 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4113 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4114 % \DeclareOption{northernkurdish}{\bbl@try@load@lang{}{kurmanji}{}}
4115 \DeclareOption{polutonikogreek}{%
4116   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}%
4117   \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4118   \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4119   \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.

```

4120 \ifx\bbl@opt@config\@nnil
4121   \@ifpackagewith{babel}{noconfigs}{}%
4122   {\InputIfFileExists{bblopts.cfg}%
4123     {\typeout{*****^J%
4124       * Local config file bblopts.cfg used^^J%
4125       *}}%
4126     {}}%
4127 \else
4128   \InputIfFileExists{\bbl@opt@config.cfg}%
4129   {\typeout{*****^J%
4130     * Local config file \bbl@opt@config.cfg used^^J%
4131     *}}%
4132   {\bbl@error{config-not-found}{}}}%
4133 \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `bbl@language@opts` are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the main language, which is processed in the third ‘main’ pass, *except* if all files are ldf *and* there is no main key. In the latter case (`\bbl@opt@main` is still `\@nnil`), the traditional way to set the main language is kept — the last loaded is the main language.

For efficiency, first preprocess the class options to remove those with `=`, which are becoming increasingly frequent (no language should contain this character).

```

4134 \def\bbl@tempf{,}
4135 \bbl@foreach\@raw@classoptionslist{%
4136   \in@{=}{#1}%
4137   \ifin@else
4138     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4139   \fi}
4140 \ifx\bbl@opt@main\@nnil
4141   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4142     \let\bbl@tempb\@empty
4143     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4144     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4145     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4146       \ifx\bbl@opt@main\@nnil % i.e., if not yet assigned
4147         \ifodd\bbl@iniflag % = *=
4148           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4149       \else % n +=

```

```

4150     \IfFileExists{#1.ldf}{\def\bbbl@opt@main{#1}}{}%
4151     \fi
4152     \fi}%
4153     \fi
4154 \else
4155     \bbbl@info{Main language set with 'main='. Except if you have\\%
4156             problems, prefer the default mechanism for setting\\%
4157             the main language, i.e., as the last declared.\\%
4158             Reported}
4159 \fi

```

A few languages are still defined explicitly. They are stored in case they are needed in the ‘main’ pass (the value can be `\relax`).

```

4160 \ifx\bbbl@opt@main\@nnil\else
4161     \bbbl@ncarg\let\bbbl@loadmain{ds@\bbbl@opt@main}%
4162     \expandafter\let\csname ds@\bbbl@opt@main\endcsname\relax
4163 \fi

```

Now define the corresponding loaders. With package options, assume the language exists. With class options, check if the option is a language by checking if the corresponding file exists.

```

4164 \bbbl@foreach\bbbl@language@opts{%
4165     \def\bbbl@tempa{#1}%
4166     \ifx\bbbl@tempa\bbbl@opt@main\else
4167         \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4168             \bbbl@ifunset{ds@#1}%
4169             {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4170             {}%
4171         \else % + * (other = ini)
4172             \DeclareOption{#1}{%
4173                 \bbbl@ldfinit
4174                 \babelprovide[@import]{#1}% %%%
4175                 \bbbl@afterldf{}}%
4176         \fi
4177     \fi}
4178 \bbbl@foreach\bbbl@tempf{%
4179     \def\bbbl@tempa{#1}%
4180     \ifx\bbbl@tempa\bbbl@opt@main\else
4181         \ifnum\bbbl@iniflag<\tw@ % 0 0 (other = ldf)
4182             \bbbl@ifunset{ds@#1}%
4183             {\IfFileExists{#1.ldf}%
4184              {\DeclareOption{#1}{\bbbl@load@language{#1}}}%
4185              {}}%
4186             {}%
4187         \else % + * (other = ini)
4188             \IfFileExists{babel-#1.tex}%
4189             {\DeclareOption{#1}{%
4190                 \bbbl@ldfinit
4191                 \babelprovide[@import]{#1}% %%%
4192                 \bbbl@afterldf{}}}%
4193             {}%
4194         \fi
4195     \fi}

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored. There is still room for last minute changes with a  $\LaTeX$  hook (not a Babel one).

The options have to be processed in the order in which the user specified them (but remember class options are processed before):

```

4196 \NewHook{babel/presets}
4197 \UseHook{babel/presets}
4198 \def\AfterBabelLanguage#1{%
4199     \bbbl@ifsamestring\CurrentOption{#1}{\global\bbbl@add\bbbl@afterlang{}}
4200 \DeclareOption*{}
4201 \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. With some options in provide, the package luatexbase is loaded (and immediately used), and therefore \babelprovide can't go inside a \DeclareOption; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate \AfterBabelLanguage.

```

4202 \bbl@trace{Option 'main'}
4203 \ifx\bbl@opt@main\@nnil
4204 \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4205 \let\bbl@tempc\@empty
4206 \edef\bbl@templ{\bbl@loaded,}
4207 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4208 \bbl@for\bbl@tempb\bbl@tempa{%
4209 \edef\bbl@tempd{\bbl@tempb,}%
4210 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4211 \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4212 \ifin@edef\bbl@tempc{\bbl@tempb}\fi}
4213 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4214 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4215 \ifx\bbl@tempb\bbl@tempc\else
4216 \bbl@warning{%
4217 Last declared language option is '\bbl@tempc',\%
4218 but the last processed one was '\bbl@tempb'.\%
4219 The main language can't be set as both a global\%
4220 and a package option. Use 'main=\bbl@tempc' as\%
4221 option. Reported}
4222 \fi
4223 \else
4224 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4225 \bbl@ldfinit
4226 \let\CurrentOption\bbl@opt@main
4227 \bbl@exp{% \bbl@opt@provide = empty if *
4228 \\\babelprovide
4229 [\bbl@opt@provide,@import,main]% %%%
4230 {\bbl@opt@main}}%
4231 \bbl@afterldf{}
4232 \DeclareOption{\bbl@opt@main}{}
4233 \else % case 0,2 (main is ldf)
4234 \ifx\bbl@loadmain\relax
4235 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4236 \else
4237 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4238 \fi
4239 \ExecuteOptions{\bbl@opt@main}
4240 \@namedef{ds@\bbl@opt@main}{}%
4241 \fi
4242 \DeclareOption*{}
4243 \ProcessOptions*
4244 \fi
4245 \bbl@exp{%
4246 \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{\begin@document}{}}}%
4247 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}}{}

```

In order to catch the case where the user didn't specify a language we check whether \bbl@main@language, has become defined. If not, the nil language is loaded.

```

4248 \ifx\bbl@main@language\undefined
4249 \bbl@info{%
4250 You haven't specified a language as a class or package\%
4251 option. I'll load 'nil'. Reported}
4252 \bbl@load@language{nil}
4253 \fi
4254 </package>

```

## 6. The kernel of Babel

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`

```
4255 (*kernel)
4256 \let\bbl@onlyswitch\@empty
4257 \input babel.def
4258 \let\bbl@onlyswitch\@undefined
4259 </kernel>
```

## 7. Error messages

They are loaded when `\bbl@error` is first called. To save space, the main code just identifies them with a tag, and messages are stored in a separate file. Since it can be loaded anywhere, you make sure some catcodes have the right value, although those for `\`, ```, `^`, `M`, `%` and `=` are reset before loading the file.

```
4260 (*errors)
4261 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4262 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4263 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4264 \catcode`\@=11 \catcode`\^=7
4265 %
4266 \ifx\MessageBreak\@undefined
4267 \gdef\bbl@error@i#1#2{%
4268 \begingroup
4269 \newlinechar=`^^J
4270 \def\{^J(babel) }%
4271 \errhelp{#2}\errmessage{\#1}%
4272 \endgroup}
4273 \else
4274 \gdef\bbl@error@i#1#2{%
4275 \begingroup
4276 \def\{\MessageBreak}%
4277 \PackageError{babel}{#1}{#2}%
4278 \endgroup}
4279 \fi
4280 \def\bbl@errmessage#1#2#3{%
4281 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4282 \bbl@error@i{#2}{#3}}
4283 % Implicit #2#3#4:
4284 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4285 %
4286 \bbl@errmessage{not-yet-available}
4287 {Not yet available}%
4288 {Find an armchair, sit down and wait}
4289 \bbl@errmessage{bad-package-option}%
4290 {Bad option '#1=#2'. Either you have misspelled the\%
4291 key or there is a previous setting of '#1'. Valid\%
4292 keys are, among others, 'shorthands', 'main', 'bidi',\%
4293 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4294 {See the manual for further details.}
4295 \bbl@errmessage{base-on-the-fly}
4296 {For a language to be defined on the fly 'base'\%
```

```

4297     is not enough, and the whole package must be\\%
4298     loaded. Either delete the 'base' option or\\%
4299     request the languages explicitly}%
4300     {See the manual for further details.}
4301 \bbl@errmessage{undefined-language}
4302     {You haven't defined the language '#1' yet.\\%
4303     Perhaps you misspelled it or your installation\\%
4304     is not complete}%
4305     {Your command will be ignored, type <return> to proceed}
4306 \bbl@errmessage{shorthand-is-off}
4307     {I can't declare a shorthand turned off (\string#2)}
4308     {Sorry, but you can't use shorthands which have been\\%
4309     turned off in the package options}
4310 \bbl@errmessage{not-a-shorthand}
4311     {The character '\string #1' should be made a shorthand character;\\%
4312     add the command \string\usesshorthands\string{#1\string} to
4313     the preamble.\\%
4314     I will ignore your instruction}%
4315     {You may proceed, but expect unexpected results}
4316 \bbl@errmessage{not-a-shorthand-b}
4317     {I can't switch '\string#2' on or off--not a shorthand}%
4318     {This character is not a shorthand. Maybe you made\\%
4319     a typing mistake? I will ignore your instruction.}
4320 \bbl@errmessage{unknown-attribute}
4321     {The attribute #2 is unknown for language #1.}%
4322     {Your command will be ignored, type <return> to proceed}
4323 \bbl@errmessage{missing-group}
4324     {Missing group for string \string#1}%
4325     {You must assign strings to some category, typically\\%
4326     captions or extras, but you set none}
4327 \bbl@errmessage{only-lua-xe}
4328     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4329     {Consider switching to these engines.}
4330 \bbl@errmessage{only-lua}
4331     {This macro is available only in LuaLaTeX}%
4332     {Consider switching to that engine.}
4333 \bbl@errmessage{unknown-provide-key}
4334     {Unknown key '#1' in \string\babelprovide}%
4335     {See the manual for valid keys}%
4336 \bbl@errmessage{unknown-mapfont}
4337     {Option '\bbl@KVP@mapfont' unknown for\\%
4338     mapfont. Use 'direction'}%
4339     {See the manual for details.}
4340 \bbl@errmessage{no-ini-file}
4341     {There is no ini file for the requested language\\%
4342     (#1: \languagename). Perhaps you misspelled it or your\\%
4343     installation is not complete}%
4344     {Fix the name or reinstall babel.}
4345 \bbl@errmessage{digits-is-reserved}
4346     {The counter name 'digits' is reserved for mapping\\%
4347     decimal digits}%
4348     {Use another name.}
4349 \bbl@errmessage{limit-two-digits}
4350     {Currently two-digit years are restricted to the\\
4351     range 0-9999}%
4352     {There is little you can do. Sorry.}
4353 \bbl@errmessage{alphabetic-too-large}
4354     {Alphabetic numeral too large (#1)}%
4355     {Currently this is the limit.}
4356 \bbl@errmessage{no-ini-info}
4357     {I've found no info for the current locale.\\%
4358     The corresponding ini file has not been loaded\\%
4359     Perhaps it doesn't exist}%

```



4360 {See the manual for details.}

4361 \bbl@errmessage{unknown-ini-field}

4362 {Unknown field '#1' in \string\BCPdata.\\%

4363 Perhaps you misspelled it}%

4364 {See the manual for details.}

4365 \bbl@errmessage{unknown-locale-key}

4366 {Unknown key for locale '#2':\\%

4367 #3\\%

4368 \string#1 will be set to \string\relax}%

4369 {Perhaps you misspelled it.}%

4370 \bbl@errmessage{adjust-only-vertical}

4371 {Currently, #1 related features can be adjusted only\\%

4372 in the main vertical list}%

4373 {Maybe things change in the future, but this is what it is.}

4374 \bbl@errmessage{layout-only-vertical}

4375 {Currently, layout related features can be adjusted only\\%

4376 in vertical mode}%

4377 {Maybe things change in the future, but this is what it is.}

4378 \bbl@errmessage{bidi-only-lua}

4379 {The bidi method 'basic' is available only in\\%

4380 luatex. I'll continue with 'bidi=default', so\\%

4381 expect wrong results}%

4382 {See the manual for further details.}

4383 \bbl@errmessage{multiple-bidi}

4384 {Multiple bidi settings inside a group}%

4385 {I'll insert a new group, but expect wrong results.}

4386 \bbl@errmessage{unknown-package-option}

4387 {Unknown option '\CurrentOption'. Either you misspelled it\\%

4388 or the language definition file \CurrentOption.ldf\\%

4389 was not found%

4390 \bbl@tempa}

4391 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%

4392 activeacute, activegrave, noconfigs, safe=, main=, math=\\%

4393 headfoot=, strings=, config=, hyphenmap=, or a language name.}

4394 \bbl@errmessage{config-not-found}

4395 {Local config file '\bbl@opt@config.cfg' not found}%

4396 {Perhaps you misspelled it.}

4397 \bbl@errmessage{late-after-babel}

4398 {Too late for \string\AfterBabelLanguage}%

4399 {Languages have been loaded, so I can do nothing}

4400 \bbl@errmessage{double-hyphens-class}

4401 {Double hyphens aren't allowed in \string\babelcharclass\\%

4402 because it's potentially ambiguous}%

4403 {See the manual for further info}

4404 \bbl@errmessage{unknown-interchar}

4405 {'#1' for '\languagename' cannot be enabled.\\%

4406 Maybe there is a typo}%

4407 {See the manual for further details.}

4408 \bbl@errmessage{unknown-interchar-b}

4409 {'#1' for '\languagename' cannot be disabled.\\%

4410 Maybe there is a typo}%

4411 {See the manual for further details.}

4412 \bbl@errmessage{charproperty-only-vertical}

4413 {\string\babelcharproperty\space can be used only in\\%

4414 vertical mode (preamble or between paragraphs)}%

4415 {See the manual for further info}

4416 \bbl@errmessage{unknown-char-property}

4417 {No property named '#2'. Allowed values are\\%

4418 direction (bc), mirror (bmg), and linebreak (lb)}%

4419 {See the manual for further info}

4420 \bbl@errmessage{bad-transform-option}

4421 {Bad option '#1' in a transform.\\%

4422 I'll ignore it but expect more errors}%

```

4423 {See the manual for further info.}
4424 \bbl@errmessage{font-conflict-transforms}
4425 {Transforms cannot be re-assigned to different\\%
4426 fonts. The conflict is in '\bbl@kv@label'.\\%
4427 Apply the same fonts or use a different label}%
4428 {See the manual for further details.}
4429 \bbl@errmessage{transform-not-available}
4430 {'#1' for '\language' cannot be enabled.\\%
4431 Maybe there is a typo or it's a font-dependent transform}%
4432 {See the manual for further details.}
4433 \bbl@errmessage{transform-not-available-b}
4434 {'#1' for '\language' cannot be disabled.\\%
4435 Maybe there is a typo or it's a font-dependent transform}%
4436 {See the manual for further details.}
4437 \bbl@errmessage{year-out-range}
4438 {Year out of range.\\%
4439 The allowed range is #1}%
4440 {See the manual for further details.}
4441 \bbl@errmessage{only-pdftex-lang}
4442 {The '#1' ldf style doesn't work with #2,\\%
4443 but you can use the ini locale instead.\\%
4444 Try adding 'provide=' to the option list. You may\\%
4445 also want to set 'bidi=' to some value}%
4446 {See the manual for further details.}
4447 \bbl@errmessage{hyphenmins-args}
4448 {\string\babelhyphenmins\ accepts either the optional\\%
4449 argument or the star, but not both at the same time}%
4450 {See the manual for further details.}
4451 </errors>
4452 <:*patterns>

```

## 8. Loading hyphenation patterns

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

```

4453 <@Make sure ProvidesFile is defined@>
4454 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4455 \xdef\bbl@format{\jobname}
4456 \def\bbl@version{<@version@>}
4457 \def\bbl@date{<@date@>}
4458 \ifx\AtBeginDocument\@undefined
4459 \def\@empty{}
4460 \fi
4461 <@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.

```

4462 \def\process@line#1#2 #3 #4 {%
4463 \ifx=#1%
4464 \process@synonym{#2}%
4465 \else
4466 \process@language{#1#2}{#3}{#4}%
4467 \fi
4468 \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.

```

4469 \toks@{}
4470 \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.

```

4471 \def\process@synonym#1{%
4472   \ifnum\last@language=\m@ne
4473     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4474   \else
4475     \expandafter\chardef\csname l@#1\endcsname\last@language
4476     \wlog{\string\l@#1=\string\language\the\last@language}%
4477     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4478       \csname\language\hyphenmins\endcsname
4479     \let\bb@elt\relax
4480     \edef\bb@languages{\bb@languages\bb@elt{#1}{\the\last@language}}}%
4481   \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 `\bb@get@enc` extracts the font encoding from the language name and stores it in `\bb@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 `\langle language \rangle 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.)

`\bb@languages` saves a snapshot of the loaded languages in the form `\bb@elt{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. 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.

```

4482 \def\process@language#1#2#3{%
4483   \expandafter\addlanguage\csname l@#1\endcsname
4484   \expandafter\language\csname l@#1\endcsname
4485   \edef\language#1}%
4486   \bb@hook@everylanguage{#1}%
4487   % > luatex
4488   \bb@get@enc#1: :@@@
4489   \begingroup
4490     \lefthyphenmin\m@ne
4491     \bb@hook@loadpatterns{#2}%
4492     % > luatex
4493     \ifnum\lefthyphenmin=\m@ne
4494     \else
4495       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4496         \the\lefthyphenmin\the\righthyphenmin}%
4497     \fi
4498   \endgroup
4499   \def\bb@tempa{#3}%

```

```

4500 \ifx\bb@tempa\@empty\else
4501   \bb@hook@loadexceptions{#3}%
4502   % > luatex
4503 \fi
4504 \let\bb@elt\relax
4505 \edef\bb@languages{%
4506   \bb@languages\bb@elt{#1}{\the\language}{#2}{\bb@tempa}}%
4507 \ifnum\the\language=\z@
4508   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4509     \set@hyphenmins\tw@\thr@@\relax
4510   \else
4511     \expandafter\expandafter\expandafter\set@hyphenmins
4512     \csname #1hyphenmins\endcsname
4513   \fi
4514   \the\toks@
4515   \toks@{ }%
4516 \fi}

```

### **\bb@get@enc**

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

```
4517 \def\bb@get@enc#1:#2:#3@@@{\def\bb@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.

```

4518 \def\bb@hook@everylanguage#1{}
4519 \def\bb@hook@loadpatterns#1{\input #1\relax}
4520 \let\bb@hook@loadexceptions\bb@hook@loadpatterns
4521 \def\bb@hook@loadkernel#1{%
4522   \def\addlanguage{\csname newlanguage\endcsname}%
4523   \def\adddialect##1##2{%
4524     \global\chardef##1##2\relax
4525     \wlog{\string##1 = a dialect from \string\language##2}}%
4526   \def\iflanguage##1{%
4527     \expandafter\ifx\csname l@##1\endcsname\relax
4528       \@nolanerr{##1}%
4529     \else
4530       \ifnum\csname l@##1\endcsname=\language
4531         \expandafter\expandafter\expandafter\@firstoftwo
4532       \else
4533         \expandafter\expandafter\expandafter\@secondoftwo
4534       \fi
4535     \fi}%
4536   \def\providehyphenmins##1##2{%
4537     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4538       \@namedef{##1hyphenmins}{##2}%
4539     \fi}%
4540   \def\set@hyphenmins##1##2{%
4541     \lefthyphenmin##1\relax
4542     \righthyphenmin##2\relax}%
4543   \def\selectlanguage{%
4544     \errhelp{Selecting a language requires a package supporting it}%
4545     \errmessage{No multilingual package has been loaded}}%
4546   \let\foreignlanguage\selectlanguage
4547   \let\otherlanguage\selectlanguage
4548   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4549   \def\bb@usehooks##1##2{% TODO. Temporary!!
4550   \def\setlocale{%
4551     \errhelp{Find an armchair, sit down and wait}%
4552     \errmessage{(babel) Not yet available}}%
4553   \let\uselocale\setlocale
4554   \let\locale\setlocale

```

```

4555 \let\selectlocale\setlocale
4556 \let\localename\setlocale
4557 \let\textlocale\setlocale
4558 \let\textlanguage\setlocale
4559 \let\languagetext\setlocale}
4560 \begingroup
4561 \def\AddBabelHook#1#2{%
4562   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4563     \def\next{\toks1}%
4564   \else
4565     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4566   \fi
4567   \next}
4568 \ifx\directlua\@undefined
4569   \ifx\XeTeXinputencoding\@undefined\else
4570     \input xebabel.def
4571   \fi
4572 \else
4573   \input luababel.def
4574 \fi
4575 \openin1 = babel-\bbl@format.cfg
4576 \ifeof1
4577 \else
4578   \input babel-\bbl@format.cfg\relax
4579 \fi
4580 \closein1
4581 \endgroup
4582 \bbl@hook@loadkernel{switch.def}

```

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

```

4583 \openin1 = language.dat

```

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

```

4584 \def\languagename{english}%
4585 \ifeof1
4586   \message{I couldn't find the file language.dat,\space
4587           I will try the file hyphen.tex}
4588   \input hyphen.tex\relax
4589   \chardef\l@english\z@
4590 \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`.

```

4591 \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.

```

4592 \loop
4593   \endlinechar\m@ne
4594   \read1 to \bbl@line
4595   \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.

```

4596   \if T\ifeof1\fi T\relax
4597   \ifx\bbl@line\@empty\else
4598     \edef\bbl@line{\bbl@line\space\space\space}%
4599     \expandafter\process@line\bbl@line\relax

```

```
4600 \fi
4601 \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.

```
4602 \begingroup
4603 \def\bbl@elt#1#2#3#4{%
4604 \global\language=#2\relax
4605 \gdef\languagename{#1}%
4606 \def\bbl@elt##1##2##3##4{}}%
4607 \bbl@languages
4608 \endgroup
4609 \fi
4610 \closeinl
```

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

```
4611 \if/\the\toks@\else
4612 \errhelp{language.dat loads no language, only synonyms}
4613 \errmessage{Orphan language synonym}
4614 \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.

```
4615 \let\bbl@line@\undefined
4616 \let\process@line@\undefined
4617 \let\process@synonym@\undefined
4618 \let\process@language@\undefined
4619 \let\bbl@get@enc@\undefined
4620 \let\bbl@hyph@enc@\undefined
4621 \let\bbl@tempa@\undefined
4622 \let\bbl@hook@loadkernel@\undefined
4623 \let\bbl@hook@everylanguage@\undefined
4624 \let\bbl@hook@loadpatterns@\undefined
4625 \let\bbl@hook@loadexceptions@\undefined
4626 </patterns>
```

Here the code for `iniTeX` ends.

## 9. luatex + xetex: common stuff

Add the bidi handler just before `luaofload`, 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 (although default also applies to `pdftex`).

```
4627 <<{*More package options}>> ≡
4628 \chardef\bbl@bidimode\z@
4629 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4630 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4631 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4632 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4633 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4634 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4635 <</More package options>>
```

**\babelfont** 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`.

```
4636 <<{*Font selection}>> ≡
4637 \bbl@trace{Font handling with fontspec}
4638 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4639 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4640 \DisableBabelHook{babel-fontspec}
```

```

4641 \@onlypreamble\babelfont
4642 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4643 \ifx\fontspec\undefined
4644 \usepackage{fontspec}%
4645 \fi
4646 \EnableBabelHook{babel-fontspec}%
4647 \edef\bb@tempa{#1}%
4648 \def\bb@tempb{#2}% Used by \bb@bbfont
4649 \bb@bbfont}
4650 \newcommand\bb@bbfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4651 \bb@ifunset{\bb@tempb family}%
4652 {\bb@providefam{\bb@tempb}}%
4653 }%
4654 % For the default font, just in case:
4655 \bb@ifunset{\bb@lsys{\languagename}}{\bb@provide@lsys{\languagename}}{}%
4656 \expandafter\bb@ifblank\expandafter{\bb@tempa}%
4657 {\bb@csarg\edef{\bb@tempb dflt@}{<#1>{#2}}% save \bb@rmdflt@
4658 \bb@exp{%
4659 \let\<bb@\bb@tempb dflt@\languagename>\<bb@\bb@tempb dflt@>%
4660 \\\bb@font@set\<bb@\bb@tempb dflt@\languagename>%
4661 \<\bb@tempb default>\<\bb@tempb family>}}%
4662 {\bb@foreach\bb@tempa{% i.e., \bb@rmdflt@lang / *sct
4663 \bb@csarg\def{\bb@tempb dflt@##1}{<#1>{#2}}}}%

```

If the family in the previous command does not exist, it must be defined. Here is how:

```

4664 \def\bb@providefam#1{%
4665 \bb@exp{%
4666 \\\newcommand\<#1default>{}% Just define it
4667 \\\bb@add@list\\bb@font@fams{#1}%
4668 \\\NewHook{#1family}%
4669 \\\DeclareRobustCommand\<#1family>{%
4670 \\\not@math@alphabet\<#1family>\relax
4671 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4672 \\\fontfamily\<#1default>%
4673 \\\UseHook{#1family}%
4674 \\\selectfont}%
4675 \\\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.

```

4676 \def\bb@nostdfont#1{%
4677 \bb@ifunset{\bb@WFF@\f@family}%
4678 {\bb@csarg\gdef{WFF@\f@family}}% Flag, to avoid dupl warns
4679 \bb@infowarn{The current font is not a babel standard family:\%
4680 #1%
4681 \fontname\font\\%
4682 There is nothing intrinsically wrong with this warning, and\\%
4683 you can ignore it altogether if you do not need these\\%
4684 families. But if they are used in the document, you should be\\%
4685 aware 'babel' will not set Script and Language for them, so\\%
4686 you may consider defining a new family with \string\babelfont.\\%
4687 See the manual for further details about \string\babelfont.\\%
4688 Reported}}
4689 {}%
4690 \gdef\bb@switchfont{%
4691 \bb@ifunset{\bb@lsys{\languagename}}{\bb@provide@lsys{\languagename}}{}%
4692 \bb@exp{% e.g., Arabic -> arabic
4693 \lowercase{\edef\\bb@tempa{\bb@c{l}{sname}}}}%
4694 \bb@foreach\bb@font@fams{%
4695 \bb@ifunset{\bb@##1dflt@\languagename}% (1) language?
4696 {\bb@ifunset{\bb@##1dflt@*\bb@tempa}% (2) from script?
4697 {\bb@ifunset{\bb@##1dflt@}% 2=F - (3) from generic?
4698 }% 123=F - nothing!
4699 {\bb@exp{% 3=T - from generic

```

```

4700     \global\let\<bbl@##1dflt@\language>%
4701         \<bbl@##1dflt@>}}}%
4702     {\bbl@exp{%          2=T - from script
4703     \global\let\<bbl@##1dflt@\language>%
4704         \<bbl@##1dflt@*\bbl@tempa>}}}%
4705     }}}%          1=T - language, already defined
4706 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4707 \bbl@foreach\bbl@font@fams{%    don't gather with prev for
4708 \bbl@ifunset{bbl@##1dflt@\language}%
4709     {\bbl@cs{famrst@##1}%
4710     \global\bbl@csarg\let{famrst@##1}\relax}%
4711     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4712     \\bbl@add\\originalTeX{%
4713     \\bbl@font@rst{\bbl@cl{##1dflt}}%
4714     \<##1default>\<##1family>{##1}}}%
4715     \\bbl@font@set\<bbl@##1dflt@\language>% the main part!
4716     \<##1default>\<##1family>}}}%
4717 \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 `\babel font`.

```

4718 \ifx\f@family\undefined\else % if latex
4719 \ifcase\bbl@engine % if pdftex
4720 \let\bbl@ckeckstdfonts\relax
4721 \else
4722 \def\bbl@ckeckstdfonts{%
4723 \begingroup
4724 \global\let\bbl@ckeckstdfonts\relax
4725 \let\bbl@tempa\@empty
4726 \bbl@foreach\bbl@font@fams{%
4727 \bbl@ifunset{bbl@##1dflt@}%
4728 {\@nameuse{##1family}}%
4729 \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4730 \bbl@exp{\\bbl@add\\bbl@tempa* \<##1family>= \f@family\\}%
4731 \space\space\fontname\font\\}%
4732 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4733 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4734 {}}%
4735 \ifx\bbl@tempa\@empty\else
4736 \bbl@infowarn{The following font families will use the default\\%
4737 settings for all or some languages:\\%
4738 \bbl@tempa
4739 There is nothing intrinsically wrong with it, but\\%
4740 'babel' will no set Script and Language, which could\\%
4741 be relevant in some languages. If your document uses\\%
4742 these families, consider redefining them with \string\babelfont.\\%
4743 Reported}%
4744 \fi
4745 \endgroup}
4746 \fi
4747 \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.

For historical reasons,  $\LaTeX$  can select two different series (bx and b), for what is conceptually a single one. This can lead to problems when a single family requires several fonts, depending on the language, mainly because ‘substitutions’ with some combinations are not done consistently – sometimes `bx/sc` is the correct font, but sometimes points to `b/n`, even if `b/sc` exists. So, some substitutions are redefined (in a somewhat hackish way, by inspecting if the variant declaration contains `>sub*`).

```

4748 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4749 \bbl@xin@{<>}{#1}%

```





**\BabelFootnote** Footnotes.

```
4800 <<{*Footnote changes}>> ≡
4801 \bbl@trace{Bidi footnotes}
4802 \ifnum\bbl@bidimode>\z@ % Any bidi=
4803 \def\bbl@footnote#1#2#3{%
4804   \@ifnextchar[%
4805     {\bbl@footnote@o{#1}{#2}{#3}}%
4806     {\bbl@footnote@x{#1}{#2}{#3}}}
4807 \long\def\bbl@footnote@x#1#2#3#4{%
4808   \bgroup
4809     \select@language@x{\bbl@main@language}%
4810     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4811   \egroup}
4812 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4813   \bgroup
4814     \select@language@x{\bbl@main@language}%
4815     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4816   \egroup}
4817 \def\bbl@footnotetext#1#2#3{%
4818   \@ifnextchar[%
4819     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4820     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4821 \long\def\bbl@footnotetext@x#1#2#3#4{%
4822   \bgroup
4823     \select@language@x{\bbl@main@language}%
4824     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4825   \egroup}
4826 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4827   \bgroup
4828     \select@language@x{\bbl@main@language}%
4829     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4830   \egroup}
4831 \def\BabelFootnote#1#2#3#4{%
4832   \ifx\bbl@fn@footnote@undefined
4833     \let\bbl@fn@footnote\footnote
4834   \fi
4835   \ifx\bbl@fn@footnotetext@undefined
4836     \let\bbl@fn@footnotetext\footnotetext
4837   \fi
4838   \bbl@ifblank{#2}%
4839     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4840     \namedef{\bbl@stripslash#1text}%
4841       {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4842   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4843     \namedef{\bbl@stripslash#1text}%
4844       {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}}
4845 \fi
4846 <</Footnote changes>>
```

## 10. Hooks for XeTeX and LuaTeX

### 10.1. XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.

Now, the code.

```
4847 <{*xetex}>
4848 \def\BabelStringsDefault{unicode}
4849 \let\xebbl@stop\relax
4850 \AddBabelHook{xetex}{encodedcommands}{%
4851   \def\bbl@tempa{#1}%
4852   \ifx\bbl@tempa@empty
```

```

4853 \XeTeXinputencoding"bytes"%
4854 \else
4855 \XeTeXinputencoding"#1"%
4856 \fi
4857 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4858 \AddBabelHook{xetex}{stopcommands}{%
4859 \xebbl@stop
4860 \let\xebbl@stop\relax}
4861 \def\bbl@input@classes{% Used in CJK intraspaces
4862 \input{load-unicode-xetex-classes.tex}%
4863 \let\bbl@input@classes\relax}
4864 \def\bbl@intraspace#1 #2 #3\@@{%
4865 \bbl@csarg\gdef{xeisp@\languagename}%
4866 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4867 \def\bbl@intrapenalty#1\@@{%
4868 \bbl@csarg\gdef{xeipn@\languagename}%
4869 {\XeTeXlinebreakpenalty #1\relax}}
4870 \def\bbl@provide@intraspace{%
4871 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4872 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4873 \ifin@
4874 \bbl@ifunset{bbl@intsp@\languagename}{}%
4875 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4876 \ifx\bbl@KVP@intraspace\@nnil
4877 \bbl@exp{%
4878 \\ \bbl@intraspace\bbl@cl{intsp}\@@}%
4879 \fi
4880 \ifx\bbl@KVP@intrapenalty\@nnil
4881 \bbl@intrapenalty0\@@
4882 \fi
4883 \fi
4884 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4885 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4886 \fi
4887 \ifx\bbl@KVP@intrapenalty\@nnil\else
4888 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4889 \fi
4890 \bbl@exp{%
4891 % TODO. Execute only once (but redundant):
4892 \\ \bbl@add<extras\languagename>{%
4893 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4894 <\bbl@xeisp@\languagename>%
4895 <\bbl@xeipn@\languagename>%
4896 \\ \bbl@tglobal<extras\languagename>%
4897 \\ \bbl@add<noextras\languagename>{%
4898 \XeTeXlinebreaklocale ""}%
4899 \\ \bbl@tglobal<noextras\languagename>%
4900 \ifx\bbl@ispacesize\undefined
4901 \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4902 \ifx\AtBeginDocument\@notprerr
4903 \expandafter\@secondoftwo % to execute right now
4904 \fi
4905 \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4906 \fi}%
4907 \fi}
4908 \ifx\DisableBabelHook\undefined\endinput\fi %%% TODO: why
4909 \let\bbl@set@renderer\relax
4910 \let\bbl@unset@renderer\relax
4911 <@Font selection@>
4912 \def\bbl@provide@extra#1{}

```

## 10.2. Support for interchar

xetex reserves some values for CJK (although they are not set in XELATEX), so we make sure they are skipped. Define some user names for the global classes, too.

```

4913 \ifnum\xe@alloc@intercharclass<\thr@@
4914 \xe@alloc@intercharclass\thr@@
4915 \fi
4916 \chardef\bbl@xe@class@default@=\z@
4917 \chardef\bbl@xe@class@cjkkideogram@=\@ne
4918 \chardef\bbl@xe@class@cjklleftpunctuation@=\tw@
4919 \chardef\bbl@xe@class@cjkrighpunctuation@=\thr@@
4920 \chardef\bbl@xe@class@boundary@=4095
4921 \chardef\bbl@xe@class@ignore@=4096

```

The machinery is activated with a hook (enabled only if actually used). Here `\bbl@tempc` is pre-set with `\bbl@usingxe@class`, defined below. The standard mechanism based on `\originalTeX` to save, set and restore values is used. `\count@` stores the previous char to be set, except at the beginning (0) and after `\bbl@upto`, which is the previous char negated, as a flag to mark a range.

```

4922 \AddBabelHook{babel-interchar}{beforeextras}{%
4923 \@nameuse{bbl@xechars@\languagename}}
4924 \DisableBabelHook{babel-interchar}
4925 \protected\def\bbl@charclass#1{%
4926 \ifnum\count@<\z@
4927 \count@-\count@
4928 \loop
4929 \bbl@exp{%
4930 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4931 \XeTeXcharclass\count@ \bbl@tempc
4932 \ifnum\count@<`#1\relax
4933 \advance\count@\@ne
4934 \repeat
4935 \else
4936 \babel@savevariable{\XeTeXcharclass`#1}%
4937 \XeTeXcharclass`#1 \bbl@tempc
4938 \fi
4939 \count@`#1\relax}

```

Now the two user macros. Char classes are declared implicitly, and then the macro to be executed at the `babel-interchar` hook is created. The list of chars to be handled by the hook defined above has internally the form `\bbl@usingxe@class\bbl@xe@class@punct@english\bbl@charclass{.}` `\bbl@charclass{,}` (etc.), where `\bbl@usingxe@class` stores the class to be applied to the subsequent characters. The `\ifcat` part deals with the alternative way to enter characters as macros (e.g., `\`). As a special case, hyphens are stored as `\bbl@upto`, to deal with ranges.

```

4940 \newcommand\bbl@ifinterchar[1]{%
4941 \let\bbl@tempa\@gobble % Assume to ignore
4942 \edef\bbl@tempb{\zap@space#1 \@empty}%
4943 \ifx\bbl@KVP@interchar\@nnil\else
4944 \bbl@replace\bbl@KVP@interchar{ }{,}%
4945 \bbl@foreach\bbl@tempb{%
4946 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
4947 \ifin@
4948 \let\bbl@tempa\@firstofone
4949 \fi}%
4950 \fi
4951 \bbl@tempa}
4952 \newcommand\IfBabelIntercharT[2]{%
4953 \bbl@carg\bbl@add{bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
4954 \newcommand\babelcharclass[3]{%
4955 \EnableBabelHook{babel-interchar}%
4956 \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
4957 \def\bbl@tempb##1{%
4958 \ifx##1\@empty\else
4959 \ifx##1-%
4960 \bbl@upto

```

```

4961     \else
4962       \bbl@charclass{%
4963         \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4964       \fi
4965       \expandafter\bbl@tempb
4966     \fi}%
4967 \bbl@ifunset{bbl@xechars@#1}%
4968   {\toks@{%
4969     \babel@savevariable\XeTeXinterchartokenstate
4970     \XeTeXinterchartokenstate\@ne
4971   }}%
4972   {\toks@\expandafter\expandafter\expandafter{%
4973     \csname bbl@xechars@#1\endcsname}}%
4974 \bbl@csarg\edef{xechars@#1}{%
4975   \the\toks@
4976   \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
4977   \bbl@tempb#3\@empty}}
4978 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
4979 \protected\def\bbl@upto{%
4980   \ifnum\count@>\z@
4981     \advance\count@\@ne
4982     \count@-\count@
4983   \else\ifnum\count@=\z@
4984     \bbl@charclass{-}%
4985   \else
4986     \bbl@error{double-hyphens-class}{#1}{#1}%
4987   \fi\fi}

```

And finally, the command with the code to be inserted. If the language doesn't define a class, then use the global one, as defined above. For the definition there is an intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<language>`.

```

4988 \def\bbl@ignoreinterchar{%
4989   \ifnum\language=\l@nohyphenation
4990     \expandafter\@gobble
4991   \else
4992     \expandafter\@firstofone
4993   \fi}
4994 \newcommand\babelinterchar[5][{}]{%
4995   \let\bbl@kv@label\@empty
4996   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
4997   \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4998   {\bbl@ignoreinterchar{#5}}%
4999   \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5000   \bbl@exp{\bbl@for\bbl@tempa{\zap@space#3 \@empty}}{%
5001     \bbl@exp{\bbl@for\bbl@tempb{\zap@space#4 \@empty}}{%
5002       \XeTeXinterchartoks
5003         \@nameuse{bbl@xeclasse@\bbl@tempa @#2}
5004         \bbl@ifunset{bbl@xeclasse@\bbl@tempa @#2}{#2} %
5005         \@nameuse{bbl@xeclasse@\bbl@tempb @#2}
5006         \bbl@ifunset{bbl@xeclasse@\bbl@tempb @#2}{#2} %
5007         = \expandafter{%
5008           \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5009           \csname\zap@space bbl@xeinter@\bbl@kv@label
5010             @#3@#4@#2 \@empty\endcsname}}}}
5011 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5012   \bbl@ifunset{bbl@ic@#1@languagename}%
5013   {\bbl@error{unknown-interchar}{#1}{#1}}%
5014   {\bbl@csarg\let{ic@#1@languagename}\@firstofone}}
5015 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5016   \bbl@ifunset{bbl@ic@#1@languagename}%
5017   {\bbl@error{unknown-interchar-b}{#1}{#1}}%
5018   {\bbl@csarg\let{ic@#1@languagename}\@gobble}}
5019 </xetex>

```

### 10.3. Layout

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 `pdftex` and `xetex`.

```
5020 (*xetex | texxet)
5021 \providecommand\bbl@provide@intraspace{}
5022 \bbl@trace{Redefinitions for bidi layout}
5023 \ifx\bbl@opt@layout\@nnil\else % if layout=. .
5024 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5025 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5026 \ifnum\bbl@bidimode>\z@ % TODO: always?
5027 \def\@hangfrom#1{%
5028   \setbox\@tempboxa\hbox{#1}}%
5029   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5030   \noindent\box\@tempboxa}
5031 \def\raggedright{%
5032   \let\\\@centercr
5033   \bbl@startskip\z@skip
5034   \@rightskip\@flushglue
5035   \bbl@endskip\@rightskip
5036   \parindent\z@
5037   \parfillskip\bbl@startskip}
5038 \def\raggedleft{%
5039   \let\\\@centercr
5040   \bbl@startskip\@flushglue
5041   \bbl@endskip\z@skip
5042   \parindent\z@
5043   \parfillskip\bbl@endskip}
5044 \fi
5045 \IfBabelLayout{lists}
5046 {\bbl@sreplace\list
5047   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5048   \def\bbl@listleftmargin{%
5049     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5050   \ifcase\bbl@engine
5051     \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
5052     \def\p@enumiii{\p@enumii}\theenumii{}%
5053   \fi
5054   \bbl@sreplace\@verbatim
5055     {\leftskip\@totalleftmargin}%
5056     {\bbl@startskip\textwidth
5057       \advance\bbl@startskip-\linewidth}%
5058   \bbl@sreplace\@verbatim
5059     {\rightskip\z@skip}%
5060     {\bbl@endskip\z@skip}}%
5061 {}
5062 \IfBabelLayout{contents}
5063 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5064   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5065 {}
5066 \IfBabelLayout{columns}
5067 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5068   \def\bbl@outputbox#1{%
5069     \hb@xt@\textwidth{%
5070       \hskip\columnwidth
5071       \hfil
5072       {\normalcolor\vrule \@width\columnseprule}%
5073       \hfil
```

```

5074     \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5075     \hskip-\textwidth
5076     \hb@xt@\columnwidth{\box\@outputbox \hss}%
5077     \hskip\columnsep
5078     \hskip\columnwidth}}}%
5079   {}
5080 <@Footnote changes@>
5081 \IfBabelLayout{footnotes}%
5082   {\BabelFootnote\footnote\languagename{}}}%
5083   \BabelFootnote\localfootnote\languagename{}}}%
5084   \BabelFootnote\mainfootnote{}}{}%
5085   {}

```

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.

```

5086 \IfBabelLayout{counters*}%
5087   {\bbl@add\bbl@opt@layout{.counters.}%
5088   \AddToHook{shipout/before}{%
5089     \let\bbl@tempa\babelsublr
5090     \let\babelsublr\@firstofone
5091     \let\bbl@save@thepage\thepage
5092     \protected@edef\thepage{\thepage}%
5093     \let\babelsublr\bbl@tempa}%
5094   \AddToHook{shipout/after}{%
5095     \let\thepage\bbl@save@thepage}}{}
5096 \IfBabelLayout{counters}%
5097   {\let\bbl@latinarabic=\@arabic
5098   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5099   \let\bbl@asciroman=\@roman
5100   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5101   \let\bbl@asciiRoman=\@Roman
5102   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5103 \fi % end if layout
5104 </xetex | texxet>

```

## 10.4. 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff. If just one encoding has been declared, then assume no switching is necessary (1).

```

5105 < *texxet>
5106 \def\bbl@provide@extra#1{%
5107   % == auto-select encoding ==
5108   \ifx\bbl@encoding@select@off\@empty\else
5109     \bbl@ifunset{\bbl@encoding@#1}%
5110     {\def\@elt##1{,##1,}%
5111     \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5112     \count@\z@
5113     \bbl@foreach\bbl@tempe{%
5114       \def\bbl@tempd{##1}% Save last declared
5115       \advance\count@\@ne}%
5116     \ifnum\count@>\@ne % (1)
5117       \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5118       \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5119       \bbl@replace\bbl@tempa{ },{,}%
5120       \global\bbl@csarg\let{encoding@#1}\@empty
5121       \bbl@xin@{,\bbl@tempd,},{,\bbl@tempa,}%
5122     \ifin@else % if main encoding included in ini, do nothing
5123       \let\bbl@tempb\relax
5124       \bbl@foreach\bbl@tempa{%
5125         \ifx\bbl@tempb\relax
5126           \bbl@xin@{,##1,},{,\bbl@tempe,}%
5127         \ifin@\def\bbl@tempb{##1}\fi
5128       \fi}%

```

```

5129         \ifx\bbbl@tempb\relax\else
5130         \bbbl@exp{%
5131         \global\<bbbl@add>\<bbbl@preextras@#1>{\<bbbl@encoding@#1>}%
5132         \gdef\<bbbl@encoding@#1>{%
5133         \\\babel@save\\\f@encoding
5134         \\\bbbl@add\\\originalTeX{\\\selectfont}%
5135         \\\fontencoding{\bbbl@tempb}%
5136         \\\selectfont}}%
5137     \fi
5138     \fi
5139     \fi}%
5140     }%
5141     \fi}
5142 </texxet>

```

## 10.5. 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, `\bbbl@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 (e.g., `\babelpatterns`).

```

5143 (*luatex)
5144 \directlua{ Babel = Babel or {} } % DL2
5145 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5146 \bbbl@trace{Read language.dat}
5147 \ifx\bbbl@readstream\undefined
5148   \csname newread\endcsname\bbbl@readstream
5149 \fi
5150 \begingroup
5151   \toks@{}
5152   \count@z@ % 0=start, 1=0th, 2=normal
5153   \def\bbbl@process@line#1#2 #3 #4 {%
5154     \ifx=#1%
5155       \bbbl@process@synonym{#2}%
5156     \else

```



```

5157     \bbl@process@language{#1#2}{#3}{#4}%
5158     \fi
5159     \ignorespaces}
5160 \def\bbl@manylang{%
5161     \ifnum\bbl@last>\@ne
5162         \bbl@info{Non-standard hyphenation setup}%
5163     \fi
5164     \let\bbl@manylang\relax}
5165 \def\bbl@process@language#1#2#3{%
5166     \ifcase\count@
5167         \@ifundefined{zth#1}{\count@tw@}{\count@\@ne}%
5168     \or
5169         \count@tw@
5170     \fi
5171     \ifnum\count@=\tw@
5172         \expandafter\addlanguage\csname l@#1\endcsname
5173         \language\allocationnumber
5174         \chardef\bbl@last\allocationnumber
5175         \bbl@manylang
5176         \let\bbl@elt\relax
5177         \xdef\bbl@languages{%
5178             \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5179     \fi
5180     \the\toks@
5181     \toks@{}}
5182 \def\bbl@process@synonym@aux#1#2{%
5183     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5184     \let\bbl@elt\relax
5185     \xdef\bbl@languages{%
5186         \bbl@languages\bbl@elt{#1}{#2}{}}}%
5187 \def\bbl@process@synonym#1{%
5188     \ifcase\count@
5189         \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5190     \or
5191         \@ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5192     \else
5193         \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5194     \fi}
5195 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5196     \chardef\l@english\z@
5197     \chardef\l@USenglish\z@
5198     \chardef\bbl@last\z@
5199     \global\@namedef{bbl@hyphendata@0}{{\hyphen.tex}}
5200     \gdef\bbl@languages{%
5201         \bbl@elt{english}{0}{\hyphen.tex}}%
5202     \bbl@elt{USenglish}{0}{}
5203 \else
5204     \global\let\bbl@languages@format\bbl@languages
5205     \def\bbl@elt#1#2#3#4{% Remove all except language 0
5206         \ifnum#2>\z@\else
5207             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5208         \fi}%
5209     \xdef\bbl@languages{\bbl@languages}%
5210 \fi
5211 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}} % Define flags
5212 \bbl@languages
5213 \openin\bbl@readstream=language.dat
5214 \ifeof\bbl@readstream
5215     \bbl@warning{I couldn't find language.dat. No additional\%
5216         patterns loaded. Reported}%
5217 \else
5218     \loop
5219         \endlinechar\m@ne

```

```

5220     \read\bbbl@readstream to \bbbl@line
5221     \endlinechar`\^^M
5222     \if T\ifeof\bbbl@readstream F\fi T\relax
5223     \ifx\bbbl@line\@empty\else
5224     \edef\bbbl@line{\bbbl@line\space\space\space}%
5225     \expandafter\bbbl@process@line\bbbl@line\relax
5226     \fi
5227   \repeat
5228 \fi
5229 \closein\bbbl@readstream
5230 \endgroup
5231 \bbbl@trace{Macros for reading patterns files}
5232 \def\bbbl@get@enc#1:#2:#3\@@@{\def\bbbl@hyph@enc{#2}}
5233 \ifx\babelcatcodetablenum\@undefined
5234 \ifx\newcatcodetable\@undefined
5235 \def\babelcatcodetablenum{5211}
5236 \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5237 \else
5238 \newcatcodetable\babelcatcodetablenum
5239 \newcatcodetable\bbbl@pattcodes
5240 \fi
5241 \else
5242 \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5243 \fi
5244 \def\bbbl@luapatterns#1#2{%
5245 \bbbl@get@enc#1::\@@@
5246 \setbox\z@\hbox\bgroup
5247 \beginingroup
5248 \savecatcodetable\babelcatcodetablenum\relax
5249 \initcatcodetable\bbbl@pattcodes\relax
5250 \catcodetable\bbbl@pattcodes\relax
5251 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5252 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5253 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5254 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5255 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5256 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5257 \input #1\relax
5258 \catcodetable\babelcatcodetablenum\relax
5259 \endgroup
5260 \def\bbbl@tempa{#2}%
5261 \ifx\bbbl@tempa\@empty\else
5262 \input #2\relax
5263 \fi
5264 \egroup}%
5265 \def\bbbl@patterns@lua#1{%
5266 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5267 \csname l@#1\endcsname
5268 \edef\bbbl@tempa{#1}%
5269 \else
5270 \csname l@#1:\f@encoding\endcsname
5271 \edef\bbbl@tempa{#1:\f@encoding}%
5272 \fi\relax
5273 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5274 \@ifundefined{bbbl@hyphendata@the\language}%
5275 {\def\bbbl@elt##1##2##3##4{%
5276 \ifnum##2=\csname l@bbbl@tempa\endcsname % #2=spanish, dutch:OT1...
5277 \def\bbbl@tempb{##3}%
5278 \ifx\bbbl@tempb\@empty\else % if not a synonymous
5279 \def\bbbl@tempc{{##3}{##4}}%
5280 \fi
5281 \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
5282 \fi}%

```

```

5283 \bbl@languages
5284 \@ifundefined{bbl@hyphendata@the\language}%
5285   {\bbl@info{No hyphenation patterns were set for\%
5286     language '\bbl@tempa'. Reported}}%
5287   {\expandafter\expandafter\expandafter\bbl@luapatterns
5288     \csname bbl@hyphendata@the\language\endcsname}}{}
5289 \endinput\fi

```

Here ends \ifx\AddBabelHook\@undefined. A few lines are only read by HYPHEN.CFG.

```

5290 \ifx\DisableBabelHook\@undefined
5291 \AddBabelHook{luatex}{everylanguage}{%
5292   \def\process@language##1##2##3{%
5293     \def\process@line####1####2 ####3 ####4 {}}
5294 \AddBabelHook{luatex}{loadpatterns}{%
5295   \input #1\relax
5296   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5297     {#{1}}}}
5298 \AddBabelHook{luatex}{loadexceptions}{%
5299   \input #1\relax
5300   \def\bbl@tempb##1##2{#{##1}#{##2}}%
5301   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5302     {\expandafter\expandafter\expandafter\bbl@tempb
5303       \csname bbl@hyphendata@the\language\endcsname}}
5304 \endinput\fi

```

Here stops reading code for HYPHEN.CFG. The following is read the 2nd time it's loaded. First, global declarations for lua.

```

5305 \begingroup % TODO - to a lua file % DL3
5306 \catcode`\%=12
5307 \catcode`\'=12
5308 \catcode`\|=12
5309 \catcode`\:=12
5310 \directlua{
5311 Babel.locale_props = Babel.locale_props or {}
5312 function Babel.lua_error(e, a)
5313 tex.print([[noexpand\csname bbl@error\endcsname]] ..
5314   e .. '{' .. (a or '') .. '}{}{')
5315 end
5316 function Babel.bytes(line)
5317   return line:gsub(".",
5318     function (chr) return unicode.utf8.char(string.byte(chr)) end)
5319 end
5320 function Babel.begin_process_input()
5321   if luatexbase and luatexbase.add_to_callback then
5322     luatexbase.add_to_callback('process_input_buffer',
5323       Babel.bytes, 'Babel.bytes')
5324   else
5325     Babel.callback = callback.find('process_input_buffer')
5326     callback.register('process_input_buffer', Babel.bytes)
5327   end
5328 end
5329 function Babel.end_process_input ()
5330   if luatexbase and luatexbase.remove_from_callback then
5331     luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5332   else
5333     callback.register('process_input_buffer', Babel.callback)
5334   end
5335 end
5336 function Babel.str_to_nodes(fn, matches, base)
5337   local n, head, last
5338   if fn == nil then return nil end
5339   for s in string.utfvalues(fn(matches)) do
5340     if base.id == 7 then
5341       base = base.replace

```

```

5342     end
5343     n = node.copy(base)
5344     n.char    = s
5345     if not head then
5346         head = n
5347     else
5348         last.next = n
5349     end
5350     last = n
5351 end
5352 return head
5353 end
5354 Babel.linebreaking = Babel.linebreaking or {}
5355 Babel.linebreaking.before = {}
5356 Babel.linebreaking.after = {}
5357 Babel.locale = {}
5358 function Babel.linebreaking.add_before(func, pos)
5359     tex.print([[noexpand\csname bbl@luaahyphenate\endcsname]])
5360     if pos == nil then
5361         table.insert(Babel.linebreaking.before, func)
5362     else
5363         table.insert(Babel.linebreaking.before, pos, func)
5364     end
5365 end
5366 function Babel.linebreaking.add_after(func)
5367     tex.print([[noexpand\csname bbl@luaahyphenate\endcsname]])
5368     table.insert(Babel.linebreaking.after, func)
5369 end
5370 function Babel.addpatterns(pp, lg)
5371     local lg = lang.new(lg)
5372     local pats = lang.patterns(lg) or ''
5373     lang.clear_patterns(lg)
5374     for p in pp:gmatch('[^%s]+') do
5375         ss = ''
5376         for i in string.utfcharacters(p:gsub('%d', '')) do
5377             ss = ss .. '%d?' .. i
5378         end
5379         ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5380         ss = ss:gsub('%.%d%?$', '%%.')
5381         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5382         if n == 0 then
5383             tex.sprint(
5384                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5385                 .. p .. [[]])
5386             pats = pats .. ' ' .. p
5387         else
5388             tex.sprint(
5389                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5390                 .. p .. [[]])
5391         end
5392     end
5393     lang.patterns(lg, pats)
5394 end
5395 Babel.characters = Babel.characters or {}
5396 Babel.ranges = Babel.ranges or {}
5397 function Babel.hlist_has_bidi(head)
5398     local has_bidi = false
5399     local ranges = Babel.ranges
5400     for item in node.traverse(head) do
5401         if item.id == node.id'glyph' then
5402             local itemchar = item.char
5403             local chardata = Babel.characters[itemchar]
5404             local dir = chardata and chardata.d or nil

```



```

5468     \fi}%
5469     \bbl@languages
5470     \@ifundefined{bbl@hyphendata@the\language}%
5471     {\bbl@info{No hyphenation patterns were set for\%
5472     language '#2'. Reported}}%
5473     {\expandafter\expandafter\expandafter\bbl@luapatterns
5474     \csname bbl@hyphendata@the\language\endcsname}}}%
5475     \@ifundefined{bbl@patterns@}{}%
5476     \begingroup
5477     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5478     \ifin@else
5479     \ifx\bbl@patterns@\@empty\else
5480     \directlua{ Babel.addpatterns(
5481     [[\bbl@patterns@]], \number\language) }%
5482     \fi
5483     \@ifundefined{bbl@patterns@#1}%
5484     \@empty
5485     {\directlua{ Babel.addpatterns(
5486     [[\space\csname bbl@patterns@#1\endcsname]],
5487     \number\language) }}%
5488     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5489     \fi
5490     \endgroup}%
5491     \bbl@exp{%
5492     \bbl@ifunset{bbl@prehc@\languagename}{}%
5493     {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5494     {\prehyphenchar=\bbl@cl{prehc}\relax}}

```

**\babelpatterns** This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@language` for language ones. We make sure there is a space between words when multiple commands are used.

```

5495 \@onlypreamble\babelpatterns
5496 \AtEndOfPackage{%
5497   \newcommand\babelpatterns[2][\@empty]{%
5498     \ifx\bbl@patterns@\relax
5499       \let\bbl@patterns@\@empty
5500     \fi
5501     \ifx\bbl@pttnlist@\@empty\else
5502       \bbl@warning{%
5503         You must not intermingle \string\selectlanguage\space and\%
5504         \string\babelpatterns\space or some patterns will not\%
5505         be taken into account. Reported}%
5506     \fi
5507     \ifx\@empty#1%
5508       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5509     \else
5510       \edef\bbl@tempb{\zap@space#1 \@empty}%
5511       \bbl@for\bbl@tempa\bbl@tempb{%
5512         \bbl@fixname\bbl@tempa
5513         \bbl@iflanguage\bbl@tempa{%
5514           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5515             \@ifundefined{bbl@patterns@\bbl@tempa}%
5516             \@empty
5517             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5518             #2}}}%
5519     \fi}}

```

## 10.6. Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

Replace regular (i.e., implicit) discretionary spaces by spaceskip, 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.

```
5520 \def\bbl@intraspace#1 #2 #3\@{%
5521 \directlua{
5522   Babel.intraspaces = Babel.intraspaces or {}
5523   Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5524     {b = #1, p = #2, m = #3}
5525   Babel.locale_props[\the\localeid].intraspace = %
5526     {b = #1, p = #2, m = #3}
5527 }}
5528 \def\bbl@intrapenalty#1\@{%
5529 \directlua{
5530   Babel.intrapenalties = Babel.intrapenalties or {}
5531   Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5532   Babel.locale_props[\the\localeid].intrapenalty = #1
5533 }}
5534 \begingroup
5535 \catcode`\%=12
5536 \catcode`\&=14
5537 \catcode`\'=12
5538 \catcode`\-=12
5539 \gdef\bbl@seaintraspace&
5540 \let\bbl@seaintraspace\relax
5541 \directlua{
5542   Babel.sea_enabled = true
5543   Babel.sea_ranges = Babel.sea_ranges or {}
5544   function Babel.set_chranges (script, chrng)
5545     local c = 0
5546     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.)%s') do
5547       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5548       c = c + 1
5549     end
5550   end
5551   function Babel.sea_disc_to_space (head)
5552     local sea_ranges = Babel.sea_ranges
5553     local last_char = nil
5554     local quad = 655360 &% 10 pt = 655360 = 10 * 65536
5555     for item in node.traverse(head) do
5556       local i = item.id
5557       if i == node.id'glyph' then
5558         last_char = item
5559       elseif i == 7 and item.subtype == 3 and last_char
5560         and last_char.char > 0x0C99 then
5561         quad = font.getfont(last_char.font).size
5562         for lg, rg in pairs(sea_ranges) do
5563           if last_char.char > rg[1] and last_char.char < rg[2] then
5564             lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5565             local intraspace = Babel.intraspaces[lg]
5566             local intrapenalty = Babel.intrapenalties[lg]
5567             local n
5568             if intrapenalty ~= 0 then
5569               n = node.new(14, 0) &% penalty
5570               n.penalty = intrapenalty
5571               node.insert_before(head, item, n)
5572             end
5573             n = node.new(12, 13) &% (glue, spaceskip)
5574             node.setglue(n, intraspace.b * quad,
5575               intraspace.p * quad,
5576               intraspace.m * quad)
5577             node.insert_before(head, item, n)
5578             node.remove(head, item)
5579           end
5580         end
5581       end
5582     end
5583   end
5584 end
```

```

5582     end
5583   end
5584 }&
5585 \bbl@luahyphenate}

```

## 10.7. 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.

```

5586 \catcode`\%=14
5587 \gdef\bbl@cjkintraspacespace{%
5588   \let\bbl@cjkintraspacespace\relax
5589   \directlua{
5590     require('babel-data-cjk.lua')
5591     Babel.cjk_enabled = true
5592     function Babel.cjk_linebreak(head)
5593       local GLYPH = node.id'glyph'
5594       local last_char = nil
5595       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5596       local last_class = nil
5597       local last_lang = nil
5598       for item in node.traverse(head) do
5599         if item.id == GLYPH then
5600           local lang = item.lang
5601           local LOCALE = node.get_attribute(item,
5602             Babel.attr_locale)
5603           local props = Babel.locale_props[LOCALE] or {}
5604           local class = Babel.cjk_class[item.char].c
5605           if props.cjk_quotes and props.cjk_quotes[item.char] then
5606             class = props.cjk_quotes[item.char]
5607           end
5608           if class == 'cp' then class = 'cl' % ) as CL
5609           elseif class == 'id' then class = 'I'
5610           elseif class == 'cj' then class = 'I' % loose
5611           end
5612           local br = 0
5613           if class and last_class and Babel.cjk_breaks[last_class][class] then
5614             br = Babel.cjk_breaks[last_class][class]
5615           end
5616           if br == 1 and props.linebreak == 'c' and
5617             lang ~= \the\l@nohyphenation\space and
5618             last_lang ~= \the\l@nohyphenation then
5619             local intrapenalty = props.intrapenalty
5620             if intrapenalty ~= 0 then
5621               local n = node.new(14, 0)      % penalty
5622               n.penalty = intrapenalty
5623               node.insert_before(head, item, n)
5624             end
5625             local intraspacespace = props.intraspacespace
5626             local n = node.new(12, 13)      % (glue, spaceskip)
5627             node.setglue(n, intraspacespace.b * quad,
5628               intraspacespace.p * quad,
5629               intraspacespace.m * quad)
5630             node.insert_before(head, item, n)
5631           end
5632           if font.getfont(item.font) then
5633             quad = font.getfont(item.font).size
5634           end
5635           last_class = class

```



```

5636         last_lang = lang
5637     else % if penalty, glue or anything else
5638         last_class = nil
5639     end
5640 end
5641 lang.hyphenate(head)
5642 end
5643 }%
5644 \bbl@luahyphenate}
5645 \gdef\bbl@luahyphenate{%
5646 \let\bbl@luahyphenate\relax
5647 \directlua{
5648     luatexbase.add_to_callback('hyphenate',
5649     function (head, tail)
5650         if Babel.linebreaking.before then
5651             for k, func in ipairs(Babel.linebreaking.before) do
5652                 func(head)
5653             end
5654         end
5655         lang.hyphenate(head)
5656         if Babel.cjk_enabled then
5657             Babel.cjk_linebreak(head)
5658         end
5659         if Babel.linebreaking.after then
5660             for k, func in ipairs(Babel.linebreaking.after) do
5661                 func(head)
5662             end
5663         end
5664         if Babel.set_hboxed then
5665             Babel.set_hboxed(head)
5666         end
5667         if Babel.sea_enabled then
5668             Babel.sea_disc_to_space(head)
5669         end
5670     end,
5671     'Babel.hyphenate')
5672 }}
5673 \endgroup
5674 \def\bbl@provide@intraspace{%
5675 \bbl@ifunset\bbl@intsp@{language}\language}%
5676 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\empty\else
5677 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5678 \ifin@ % cjk
5679 \bbl@cjk@intraspace
5680 \directlua{
5681     Babel.locale_props = Babel.locale_props or {}
5682     Babel.locale_props[\the\localeid].linebreak = 'c'
5683 }%
5684 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5685 \ifx\bbl@KVP@intrapenalty\@nnil
5686 \bbl@intrapenalty0\@@
5687 \fi
5688 \else % sea
5689 \bbl@sea@intraspace
5690 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5691 \directlua{
5692     Babel.sea_ranges = Babel.sea_ranges or {}
5693     Babel.set_chranges('\bbl@cl{sbcpr}',
5694     '\bbl@cl{chrng}')
5695 }%
5696 \ifx\bbl@KVP@intrapenalty\@nnil
5697 \bbl@intrapenalty0\@@
5698 \fi

```

```

5699     \fi
5700     \fi
5701     \ifx\bbl@KVP@intrapenalty\@nnil\else
5702     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5703     \fi}}

```

## 10.8. Arabic justification

WIP. `\bbl@arabicjust` is executed with both elongated and kashida. This must be fine tuned. The attribute `kashida` is set by transforms with `kashida`.

```

5704 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5705 \def\bblar@chars{%
5706   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5707   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5708   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5709 \def\bblar@elongated{%
5710   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5711   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5712   0649,064A}
5713 \beginingroup
5714   \catcode\_ =11 \catcode\_ :=11
5715   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5716 \endgroup
5717 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5718   \let\bbl@arabicjust\relax
5719   \newattribute\bblar@kashida
5720   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5721   \bblar@kashida=\z@
5722   \bbl@patchfont{\bbl@parsejalt}}%
5723   \directlua{
5724     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5725     Babel.arabic.elong_map[\the\localeid] = {}
5726     luatexbase.add_to_callback('post_linebreak_filter',
5727       Babel.arabic.justify, 'Babel.arabic.justify')
5728     luatexbase.add_to_callback('hpack_filter',
5729       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5730   }}%

```

Save both node lists to make replacement. TODO. Save also widths to make computations.

```

5731 \def\bblar@fetchjalt#1#2#3#4{%
5732   \bbl@exp{\bbl@foreach#1}{%
5733     \bbl@ifunset\bblar@JE@##1}%
5734     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5735     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5736   \directlua{%
5737     local last = nil
5738     for item in node.traverse(tex.box[0].head) do
5739       if item.id == node.id'glyph' and item.char > 0x600 and
5740         not (item.char == 0x200D) then
5741         last = item
5742       end
5743     end
5744     Babel.arabic.#3['##1#4'] = last.char
5745   }}

```

Elongated forms. Brute force. No rules at all, yet. The ideal: look at `jalt` table. And perhaps other tables (`falt?`, `csw?`). What about `kaf`? And diacritic positioning?

```

5746 \gdef\bbl@parsejalt{%
5747   \ifx\addfontfeature\undefined\else
5748     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5749     \ifin@
5750     \directlua{%
5751       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then

```

```

5752         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5753         tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5754     end
5755     }%
5756     \fi
5757 \fi}
5758 \gdef\bbl@parsejalti{%
5759 \beginingroup
5760 \let\bbl@parsejalt\relax % To avoid infinite loop
5761 \edef\bbl@tempb{\fontid\font}%
5762 \bblar@nofswarn
5763 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5764 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5765 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5766 \addfontfeature{RawFeature+=jalt}%
5767 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5768 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5769 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5770 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5771 \directlua{%
5772     for k, v in pairs(Babel.arabic.from) do
5773         if Babel.arabic.dest[k] and
5774             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5775             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5776                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5777         end
5778     end
5779 }%
5780 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5781 \beginingroup
5782 \catcode`#=11
5783 \catcode`~ =11
5784 \directlua{
5785
5786 Babel.arabic = Babel.arabic or {}
5787 Babel.arabic.from = {}
5788 Babel.arabic.dest = {}
5789 Babel.arabic.justify_factor = 0.95
5790 Babel.arabic.justify_enabled = true
5791 Babel.arabic.kashida_limit = -1
5792
5793 function Babel.arabic.justify(head)
5794     if not Babel.arabic.justify_enabled then return head end
5795     for line in node.traverse_id(node.id'hlist', head) do
5796         Babel.arabic.justify_hlist(head, line)
5797     end
5798     return head
5799 end
5800
5801 function Babel.arabic.justify_hbox(head, gc, size, pack)
5802     local has_inf = false
5803     if Babel.arabic.justify_enabled and pack == 'exactly' then
5804         for n in node.traverse_id(12, head) do
5805             if n.stretch_order > 0 then has_inf = true end
5806         end
5807         if not has_inf then
5808             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5809         end
5810     end
5811     return head
5812 end

```

```

5813
5814 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5815     local d, new
5816     local k_list, k_item, pos_inline
5817     local width, width_new, full, k_curr, wt_pos, goal, shift
5818     local subst_done = false
5819     local elong_map = Babel.arabic.elong_map
5820     local cnt
5821     local last_line
5822     local GLYPH = node.id'glyph'
5823     local KASHIDA = Babel.attr_kashida
5824     local LOCALE = Babel.attr_locale
5825
5826     if line == nil then
5827         line = {}
5828         line.glue_sign = 1
5829         line.glue_order = 0
5830         line.head = head
5831         line.shift = 0
5832         line.width = size
5833     end
5834
5835     % Exclude last line. todo. But-- it discards one-word lines, too!
5836     % ? Look for glue = 12:15
5837     if (line.glue_sign == 1 and line.glue_order == 0) then
5838         elongs = {}      % Stores elongated candidates of each line
5839         k_list = {}      % And all letters with kashida
5840         pos_inline = 0   % Not yet used
5841
5842         for n in node.traverse_id(GLYPH, line.head) do
5843             pos_inline = pos_inline + 1 % To find where it is. Not used.
5844
5845             % Elongated glyphs
5846             if elong_map then
5847                 local locale = node.get_attribute(n, LOCALE)
5848                 if elong_map[locale] and elong_map[locale][n.font] and
5849                     elong_map[locale][n.font][n.char] then
5850                     table.insert(elongs, {node = n, locale = locale} )
5851                     node.set_attribute(n.prev, KASHIDA, 0)
5852                 end
5853             end
5854
5855             % Tatwil. First create a list of nodes marked with kashida. The
5856             % rest of nodes can be ignored. The list of used weights is build
5857             % when transforms with the key kashida= are declared.
5858             if Babel.kashida_wts then
5859                 local k_wt = node.get_attribute(n, KASHIDA)
5860                 if k_wt > 0 then % todo. parameter for multi inserts
5861                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5862                 end
5863             end
5864
5865             end % of node.traverse_id
5866
5867             if #elongs == 0 and #k_list == 0 then goto next_line end
5868             full = line.width
5869             shift = line.shift
5870             goal = full * Babel.arabic.justify_factor % A bit crude
5871             width = node.dimensions(line.head) % The 'natural' width
5872
5873             % == Elongated ==
5874             % Original idea taken from 'chickenize'
5875             while (#elongs > 0 and width < goal) do

```

```

5876     subst_done = true
5877     local x = #elongs
5878     local curr = elongs[x].node
5879     local oldchar = curr.char
5880     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5881     width = node.dimensions(line.head) % Check if the line is too wide
5882     % Substitute back if the line would be too wide and break:
5883     if width > goal then
5884         curr.char = oldchar
5885         break
5886     end
5887     % If continue, pop the just substituted node from the list:
5888     table.remove(elongs, x)
5889 end
5890
5891 % == Tatwil ==
5892 % Traverse the kashida node list so many times as required, until
5893 % the line is filled. The first pass adds a tatweel after each
5894 % node with kashida in the line, the second pass adds another one,
5895 % and so on. In each pass, add first the kashida with the highest
5896 % weight, then with lower weight and so on.
5897 if #k_list == 0 then goto next_line end
5898
5899 width = node.dimensions(line.head) % The 'natural' width
5900 k_curr = #k_list % Traverse backwards, from the end
5901 wt_pos = 1
5902
5903 while width < goal do
5904     subst_done = true
5905     k_item = k_list[k_curr].node
5906     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5907         d = node.copy(k_item)
5908         d.char = 0x0640
5909         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5910         d.xoffset = 0
5911         line.head, new = node.insert_after(line.head, k_item, d)
5912         width_new = node.dimensions(line.head)
5913         if width > goal or width == width_new then
5914             node.remove(line.head, new) % Better compute before
5915             break
5916         end
5917         if Babel.fix_diacr then
5918             Babel.fix_diacr(k_item.next)
5919         end
5920         width = width_new
5921     end
5922     if k_curr == 1 then
5923         k_curr = #k_list
5924         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5925     else
5926         k_curr = k_curr - 1
5927     end
5928 end
5929
5930 % Limit the number of tatweel by removing them. Not very efficient,
5931 % but it does the job in a quite predictable way.
5932 if Babel.arabic.kashida_limit > -1 then
5933     cnt = 0
5934     for n in node.traverse_id(GLYPH, line.head) do
5935         if n.char == 0x0640 then
5936             cnt = cnt + 1
5937             if cnt > Babel.arabic.kashida_limit then
5938                 node.remove(line.head, n)

```

```

5939         end
5940     else
5941         cnt = 0
5942     end
5943 end
5944 end
5945
5946 ::next_line::
5947
5948 % Must take into account marks and ins, see luatex manual.
5949 % Have to be executed only if there are changes. Investigate
5950 % what's going on exactly.
5951 if subst_done and not gc then
5952     d = node.hpack(line.head, full, 'exactly')
5953     d.shift = shift
5954     node.insert_before(head, line, d)
5955     node.remove(head, line)
5956 end
5957 end % if process line
5958 end
5959 }
5960 \endgroup
5961 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

## 10.9. Common stuff

First, a couple of auxiliary macros to set the renderer according to the script. This is done by patching temporarily the low-level fontspec macro containing the current features set with `\defaultfontfeatures`. Admittedly this is somewhat dangerous, but that way the latter command still works as expected, because the renderer is set just before other settings. In xetex they are set to `\relax`.

```

5962 \def\bbl@scr@node@list{%
5963   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
5964   ,Greek,Latin,Old Church Slavonic Cyrillic,}
5965 \ifnum\bbl@bidimode=102 % bidi-r
5966   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
5967 \fi
5968 \def\bbl@set@renderer{%
5969   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
5970   \ifin@
5971     \let\bbl@unset@renderer\relax
5972   \else
5973     \bbl@exp{%
5974       \def\\bbl@unset@renderer{%
5975         \def<g__fontspec_default_fontopts_clist>{%
5976           \[g__fontspec_default_fontopts_clist]}%
5977         \def<g__fontspec_default_fontopts_clist>{%
5978           Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}%
5979       \fi}
5980 <@Font selection@>

```

## 10.10. 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 the function `Babel.locale_map`, which just traverse the node list to carry out the replacements. The table `loc_to_scr` stores the script range for each locale (whose id is the key), copied from this table (so that it can be modified on a locale basis); there is an intermediate table named `chr_to_loc` built on the fly for optimization, which maps a char to the locale. This locale is then used to get the `\language` 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.

```

5981 % TODO - to a lua file

```

```

5982 \directlua{% DL6
5983 Babel.script_blocks = {
5984   ['dflt'] = {},
5985   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5986             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5987   ['Armn'] = {{0x0530, 0x058F}},
5988   ['Beng'] = {{0x0980, 0x09FF}},
5989   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5990   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5991   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5992             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5993   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5994   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5995             {0xAB00, 0xAB2F}},
5996   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5997   % Don't follow strictly Unicode, which places some Coptic letters in
5998   % the 'Greek and Coptic' block
5999   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6000   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6001             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6002             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6003             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6004             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6005             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6006   ['Hebr'] = {{0x0590, 0x05FF}},
6007   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6008             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6009   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6010   ['Knda'] = {{0x0C80, 0x0CFF}},
6011   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6012             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6013             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6014   ['Lao'] = {{0x0E80, 0x0EFF}},
6015   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6016             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6017             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6018   ['Mahj'] = {{0x11150, 0x1117F}},
6019   ['Mlym'] = {{0x0D00, 0x0D7F}},
6020   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6021   ['Orya'] = {{0x0B00, 0x0B7F}},
6022   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6023   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6024   ['Taml'] = {{0x0B80, 0x0BFF}},
6025   ['Telu'] = {{0x0C00, 0x0C7F}},
6026   ['Tfng'] = {{0x2D30, 0x2D7F}},
6027   ['Thai'] = {{0x0E00, 0x0E7F}},
6028   ['Tibt'] = {{0x0F00, 0x0FFF}},
6029   ['Vaii'] = {{0xA500, 0xA63F}},
6030   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6031 }
6032
6033 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6034 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6035 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6036
6037 function Babel.locale_map(head)
6038   if not Babel.locale_mapped then return head end
6039
6040   local LOCALE = Babel.attr_locale
6041   local GLYPH = node.id('glyph')
6042   local inmath = false
6043   local toloc_save
6044   for item in node.traverse(head) do

```

```

6045 local toloc
6046 if not inmath and item.id == GLYPH then
6047   % Optimization: build a table with the chars found
6048   if Babel.chr_to_loc[item.char] then
6049     toloc = Babel.chr_to_loc[item.char]
6050   else
6051     for lc, maps in pairs(Babel.loc_to_scr) do
6052       for _, rg in pairs(maps) do
6053         if item.char >= rg[1] and item.char <= rg[2] then
6054           Babel.chr_to_loc[item.char] = lc
6055           toloc = lc
6056           break
6057         end
6058       end
6059     end
6060     % Treat composite chars in a different fashion, because they
6061     % 'inherit' the previous locale.
6062     if (item.char >= 0x0300 and item.char <= 0x036F) or
6063        (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6064        (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6065       Babel.chr_to_loc[item.char] = -2000
6066       toloc = -2000
6067     end
6068     if not toloc then
6069       Babel.chr_to_loc[item.char] = -1000
6070     end
6071   end
6072   if toloc == -2000 then
6073     toloc = toloc_save
6074   elseif toloc == -1000 then
6075     toloc = nil
6076   end
6077   if toloc and Babel.locale_props[toloc] and
6078      Babel.locale_props[toloc].letters and
6079      tex.getcatcode(item.char) \string== 11 then
6080     toloc = nil
6081   end
6082   if toloc and Babel.locale_props[toloc].script
6083      and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6084      and Babel.locale_props[toloc].script ==
6085      Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6086     toloc = nil
6087   end
6088   if toloc then
6089     if Babel.locale_props[toloc].lg then
6090       item.lang = Babel.locale_props[toloc].lg
6091       node.set_attribute(item, LOCALE, toloc)
6092     end
6093     if Babel.locale_props[toloc]['/'..item.font] then
6094       item.font = Babel.locale_props[toloc]['/'..item.font]
6095     end
6096   end
6097   toloc_save = toloc
6098 elseif not inmath and item.id == 7 then % Apply recursively
6099   item.replace = item.replace and Babel.locale_map(item.replace)
6100   item.pre     = item.pre and Babel.locale_map(item.pre)
6101   item.post    = item.post and Babel.locale_map(item.post)
6102 elseif item.id == node.id'math' then
6103   inmath = (item.subtype == 0)
6104 end
6105 end
6106 return head
6107 end

```



6108 }

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

```
6109 \newcommand\babelcharproperty[1]{%
6110   \count@=#1\relax
6111   \ifvmode
6112     \expandafter\babel@chprop
6113   \else
6114     \babel@error{charproperty-only-vertical}{#1}{#1}%
6115   \fi}
6116 \newcommand\babel@chprop[3][\the\count@]{%
6117   \@tempcnta=#1\relax
6118   \babel@ifunset{babel@chprop@#2}% {unknown-char-property}
6119     {\babel@error{unknown-char-property}{#2}{#2}}%
6120   }%
6121   \loop
6122     \babel@cs{chprop@#2}{#3}%
6123   \ifnum\count@<\@tempcnta
6124     \advance\count@\@ne
6125   \repeat}
6126 \def\babel@chprop@direction#1{%
6127   \directlua{
6128     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6129     Babel.characters[\the\count@]['d'] = '#1'
6130   }}
6131 \let\babel@chprop@bc\babel@chprop@direction
6132 \def\babel@chprop@mirror#1{%
6133   \directlua{
6134     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6135     Babel.characters[\the\count@]['m'] = '\number#1'
6136   }}
6137 \let\babel@chprop@bmg\babel@chprop@mirror
6138 \def\babel@chprop@linebreak#1{%
6139   \directlua{
6140     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6141     Babel.cjk_characters[\the\count@]['c'] = '#1'
6142   }}
6143 \let\babel@chprop@lb\babel@chprop@linebreak
6144 \def\babel@chprop@locale#1{%
6145   \directlua{
6146     Babel.chr_to_loc = Babel.chr_to_loc or {}
6147     Babel.chr_to_loc[\the\count@] =
6148       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
6149   }}
```

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.

```
6150 \directlua{% DL7
6151   Babel.nohyphenation = \the\l@nohyphenation
6152 }
```

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, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```
6153 \begingroup
6154 \catcode`\~ = 12
6155 \catcode`\% = 12
```

```

6156 \catcode`\&=14
6157 \catcode`\|=12
6158 \gdef\babelprehyphenation{&%
6159 \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6160 \gdef\babelposthyphenation{&%
6161 \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6162 \gdef\bbl@settransform#1[#2]#3#4#5{&%
6163 \ifcase#1
6164 \bbl@activateprehyphen
6165 \or
6166 \bbl@activateposthyphen
6167 \fi
6168 \begingroup
6169 \def\babeltempa{\bbl@add@list\babeltempb}&%
6170 \let\babeltempb\@empty
6171 \def\bbl@tempa{#5}&%
6172 \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
6173 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6174 \bbl@ifsamestring{##1}{remove}&%
6175 {\bbl@add@list\babeltempb{nil}}&%
6176 {\directlua{
6177 local rep = [=##1]=]
6178 local three_args = '%s*=%s*([%-d%.%a{|}]+)%s+([%-d%.%a{|}]+)%s+([%-d%.%a{|}]+)'
6179 &% Numeric passes directly: kern, penalty...
6180 rep = rep:gsub('^%s*(remove)%s*$','remove = true')
6181 rep = rep:gsub('^%s*(insert)%s*','insert = true, ')
6182 rep = rep:gsub('^%s*(after)%s*','after = true, ')
6183 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6184 rep = rep:gsub('node%s*=%s*(%a+)%s*(%a*)', Babel.capture_node)
6185 rep = rep:gsub(' (norule)' .. three_args,
6186 'norule = { ' .. '%2, %3, %4' .. ' }')
6187 if #1 == 0 or #1 == 2 then
6188 rep = rep:gsub(' (space)' .. three_args,
6189 'space = { ' .. '%2, %3, %4' .. ' }')
6190 rep = rep:gsub(' (spacefactor)' .. three_args,
6191 'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6192 rep = rep:gsub(' (kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6193 &% Transform values
6194 rep, n = rep:gsub( '{([%a%-%.]+)|([%a%_%.]+)}',
6195 function(v,d)
6196 return string.format (
6197 '\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6198 v,
6199 load( 'return Babel.locale_props'..'
6200 '\the\csname bbl@id@@#3\endcsname'..' d)() )
6201 end )
6202 rep, n = rep:gsub( '{([%a%-%.]+)|([%-d%.]+)}',
6203 '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6204 end
6205 if #1 == 1 then
6206 rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6207 rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6208 rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6209 end
6210 tex.print([[ \string\babeltempa{[]} .. rep .. [{}]])
6211 }}&%
6212 \bbl@foreach\babeltempb{&%
6213 \bbl@forkv{##1}{&%
6214 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6215 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6216 \ifin\else
6217 \bbl@error{bad-transform-option}{###1}{}&%
6218 \fi}}&%

```

```

6219 \let\bbl@kv@attribute\relax
6220 \let\bbl@kv@label\relax
6221 \let\bbl@kv@fonts\empty
6222 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6223 \ifx\bbl@kv@fonts\empty\else\bbl@settransform\fi
6224 \ifx\bbl@kv@attribute\relax
6225   \ifx\bbl@kv@label\relax\else
6226     \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6227     \bbl@replace\bbl@kv@fonts{ }{,}&%
6228     \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6229     \count@ \z@
6230     \def\bbl@elt##1##2##3{&%
6231       \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6232       {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6233         {\count@\@ne}&%
6234         {\bbl@error{font-conflict-transforms}{}}}&%
6235       {}}&%
6236     \bbl@transform@list
6237     \ifnum\count@=\z@
6238       \bbl@exp{\global\bbl@add\bbl@transform@list
6239         {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6240     \fi
6241     \bbl@ifunset{\bbl@kv@attribute}&%
6242     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6243     {}&%
6244     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6245   \fi
6246 \else
6247   \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6248 \fi
6249 \directlua{
6250   local lbr = Babel.linebreaking.replacements[#1]
6251   local u = unicode.utf8
6252   local id, attr, label
6253   if #1 == 0 then
6254     id = \the\csname bbl@id@#3\endcsname\space
6255   else
6256     id = \the\csname l@#3\endcsname\space
6257   end
6258   \ifx\bbl@kv@attribute\relax
6259     attr = -1
6260   \else
6261     attr = luatexbase.registernumber'\bbl@kv@attribute'
6262   \fi
6263   \ifx\bbl@kv@label\relax\else &% Same refs:
6264     label = [==[\bbl@kv@label]==]
6265   \fi
6266   &% Convert pattern:
6267   local patt = string.gsub([==[#4]==], '%s', '')
6268   if #1 == 0 then
6269     patt = string.gsub(patt, '|', ' ')
6270   end
6271   if not u.find(patt, '()', nil, true) then
6272     patt = '()' .. patt .. '()'
6273   end
6274   if #1 == 1 then
6275     patt = string.gsub(patt, '%(%)%', '^()')
6276     patt = string.gsub(patt, '%$(%)%', '()$')
6277   end
6278   patt = u.gsub(patt, '{(.)}',
6279     function (n)
6280       return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6281     end)

```

```

6282     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6283         function (n)
6284             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6285         end)
6286     lbkr[id] = lbkr[id] or {}
6287     table.insert(lbkr[id],
6288         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6289 }&%
6290 \endgroup}
6291 \endgroup
6292 \let\bbl@transfont@list\@empty
6293 \def\bbl@settransfont{%
6294     \global\let\bbl@settransfont\relax % Execute only once
6295     \gdef\bbl@transfont{%
6296         \def\bbl@elt####1####2####3{%
6297             \bbl@ifblank{####3}%
6298                 {\count@tw@}% Do nothing if no fonts
6299                 {\count@z@
6300                 \bbl@vforeach{####3}{%
6301                     \def\bbl@tempd{#####1}%
6302                     \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6303                     \ifx\bbl@tempd\bbl@tempe
6304                         \count@\@ne
6305                     \else\ifx\bbl@tempd\bbl@transfam
6306                         \count@\@ne
6307                     \fi\fi}%
6308                 \ifcase\count@
6309                     \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6310                 \or
6311                     \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6312                 \fi}}%
6313         \bbl@transfont@list}%
6314     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6315     \gdef\bbl@transfam{-unknown-}%
6316     \bbl@foreach\bbl@font@fams{%
6317         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6318         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6319         {\xdef\bbl@transfam{##1}}%
6320     {}}}
6321 \DeclareRobustCommand\enablelocaletransform[1]{%
6322     \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6323     {\bbl@error{transform-not-available}{#1}{}}}%
6324     {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6325 \DeclareRobustCommand\disablelocaletransform[1]{%
6326     \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6327     {\bbl@error{transform-not-available-b}{#1}{}}}%
6328     {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6329 \def\bbl@activateposthyphen{%
6330     \let\bbl@activateposthyphen\relax
6331     \ifx\bbl@attr@hboxed\@undefined
6332         \newattribute\bbl@attr@hboxed
6333     \fi
6334     \directlua{
6335         require('babel-transforms.lua')
6336         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6337     }}
6338 \def\bbl@activateprehyphen{%
6339     \let\bbl@activateprehyphen\relax
6340     \ifx\bbl@attr@hboxed\@undefined
6341         \newattribute\bbl@attr@hboxed
6342     \fi
6343     \directlua{
6344         require('babel-transforms.lua')}

```

```

6345   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6346   }}
6347 \newcommand\SetTransformValue[3]{%
6348   \directlua{
6349     Babel.locale_props[\the\csname bbl@id@@#1\endcsname].vars["#2"] = #3
6350   }}

```

The following experimental (and unfinished) macro applies the prehyphenation transforms for the current locale to a string (characters and spaces) and processes it in a fully expandable way (among other limitations, the string can't contain ]==]). The way it operates is admittedly rather cumbersome: it converts the string to a node list, processes it, and converts it back to a string. The lua code is in the lua file below.

```

6351 \newcommand\localeprehyphenation[1]{%
6352   \directlua{ Babel.string_prehyphenation(=[=#1]=], \the\localeid) }}

```

## 10.11.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  $\TeX$ . Just in case, consider the possibility it has not been loaded.

```

6353 \def\bbl@activate@preotf{%
6354   \let\bbl@activate@preotf\relax % only once
6355   \directlua{
6356     function Babel.pre_otfload_v(head)
6357       if Babel.numbers and Babel.digits_mapped then
6358         head = Babel.numbers(head)
6359       end
6360       if Babel.bidi_enabled then
6361         head = Babel.bidi(head, false, dir)
6362       end
6363       return head
6364     end
6365     %
6366     function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6367       if Babel.numbers and Babel.digits_mapped then
6368         head = Babel.numbers(head)
6369       end
6370       if Babel.bidi_enabled then
6371         head = Babel.bidi(head, false, dir)
6372       end
6373       return head
6374     end
6375     %
6376     luatexbase.add_to_callback('pre_linebreak_filter',
6377       Babel.pre_otfload_v,
6378       'Babel.pre_otfload_v',
6379     luatexbase.priority_in_callback('pre_linebreak_filter',
6380       'luaotfload.node_processor') or nil)
6381     %
6382     luatexbase.add_to_callback('hpack_filter',
6383       Babel.pre_otfload_h,
6384       'Babel.pre_otfload_h',
6385     luatexbase.priority_in_callback('hpack_filter',
6386       'luaotfload.node_processor') or nil)
6387   }}

```

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=`. The hack for the PUA is no longer necessary with basic (24.8), but it's kept in `basic-r`.

```

6388 \breakafterdirmode=1
6389 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6390   \let\bbl@beforeforeign\leavevmode

```

```

6391 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6392 \RequirePackage{luatexbase}
6393 \bbl@activate@preotf
6394 \directlua{
6395   require('babel-data-bidi.lua')
6396   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6397     require('babel-bidi-basic.lua')
6398   \or
6399     require('babel-bidi-basic-r.lua')
6400     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6401     table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6402     table.insert(Babel.ranges, {0x10000, 0x10FFFD, 'on'})
6403   \fi}
6404 \newattribute\bbl@attr@dir
6405 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6406 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6407 \fi
6408 \chardef\bbl@thetextdir\z@
6409 \chardef\bbl@thepardir\z@
6410 \def\bbl@getluadir#1{%
6411   \directlua{
6412     if tex.#ldir == 'TLT' then
6413       tex.sprint('0')
6414     elseif tex.#ldir == 'TRT' then
6415       tex.sprint('1')
6416     else
6417       tex.sprint('0')
6418     end}}
6419 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/l rl
6420   \ifcase#3\relax
6421     \ifcase\bbl@getluadir{#1}\relax\else
6422       #2 TLT\relax
6423     \fi
6424   \else
6425     \ifcase\bbl@getluadir{#1}\relax
6426       #2 TRT\relax
6427     \fi
6428   \fi}
6429 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6430 \def\bbl@thedir{0}
6431 \def\bbl@textdir#1{%
6432   \bbl@setluadir{text}\textdir{#1}%
6433   \chardef\bbl@thetextdir#1\relax
6434   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6435   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6436 \def\bbl@pardir#1{% Used twice
6437   \bbl@setluadir{par}\pardir{#1}%
6438   \chardef\bbl@thepardir#1\relax}
6439 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6440 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6441 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

RTL text inside math needs special attention. It affects not only to actual math stuff, but also to
'tabular', which is based on a fake math.

6442 \ifnum\bbl@bidimode>\z@ % Any bidi=
6443   \def\bbl@insidemath{0}%
6444   \def\bbl@everymath{\def\bbl@insidemath{1}}
6445   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6446   \frozen@everymath\expandafter{%
6447     \expandafter\bbl@everymath\the\frozen@everymath}
6448   \frozen@everydisplay\expandafter{%
6449     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6450 \AtBeginDocument{

```

```

6451 \directlua{
6452     function Babel.math_box_dir(head)
6453         if not (token.get_macro('bbl@insidemath') == '0') then
6454             if Babel.hlist_has_bidi(head) then
6455                 local d = node.new(node.id'dir')
6456                 d.dir = '+TRT'
6457                 node.insert_before(head, node.has_glyph(head), d)
6458                 local inmath = false
6459                 for item in node.traverse(head) do
6460                     if item.id == 11 then
6461                         inmath = (item.subtype == 0)
6462                     elseif not inmath then
6463                         node.set_attribute(item,
6464                             Babel.attr_dir, token.get_macro('bbl@thedir'))
6465                     end
6466                 end
6467             end
6468         end
6469         return head
6470     end
6471     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6472         "Babel.math_box_dir", 0)
6473     if Babel.unset_atdir then
6474         luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6475             "Babel.unset_atdir")
6476         luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6477             "Babel.unset_atdir")
6478     end
6479 } }%
6480 \fi

```

Experimental. Tentative name.

```

6481 \DeclareRobustCommand\localebox[1]{%
6482     {\def\bbl@insidemath{0}%
6483         \mbox{\foreignlanguage{\language}\#1}}}

```

## 10.12 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.

Still, there are three areas deserving special attention, namely, tabular, math, and graphics, text and intrinsically left-to-right elements are intermingled. I've made some progress in graphics, but they're essentially hacks; I've also made some progress in 'tabular', but when I decided to tackle math (both standard math and 'amsmath') the nightmare began. I'm still not sure how 'amsmath' should be modified, but the main problem is that, boxes are "generic" containers that can hold text, math, and graphics (even at the same time; remember that inline math is included in the list of text nodes marked with 'math' (11) nodes too).

`\@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.

```

6484 \bbl@trace{Redefinitions for bidi layout}
6485 %
6486 <<{*More package options}>> ≡
6487 \chardef\bbl@eqnpos\z@
6488 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6489 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}

```

```

6490 <</More package options>>
6491 %
6492 \ifnum\bb@bidimode>\z@ % Any bidi=
6493 \matheqdirmode\@ne % A luatex primitive
6494 \let\bb@eqnodir\relax
6495 \def\bb@eqdel{()}
6496 \def\bb@eqnum{%
6497   {\normalfont\normalcolor
6498     \expandafter\@firstoftwo\bb@eqdel
6499     \theequation
6500     \expandafter\@secondoftwo\bb@eqdel}}
6501 \def\bb@puteqno#1{\eqno\hbox{#1}}
6502 \def\bb@putleqno#1{\leqno\hbox{#1}}
6503 \def\bb@eqno@flip#1{%
6504   \ifdim\predisplaysize=-\maxdimen
6505     \eqno
6506     \hb@xt@.01pt{%
6507       \hb@xt@\displaywidth{\hss{#1}\glet\bb@upset\@currentlabel}}\hss}%
6508   \else
6509     \leqno\hbox{#1}\glet\bb@upset\@currentlabel}%
6510   \fi
6511   \bb@exp{\def\\\@currentlabel{\[bb@upset]}}
6512 \def\bb@leqno@flip#1{%
6513   \ifdim\predisplaysize=-\maxdimen
6514     \leqno
6515     \hb@xt@.01pt{%
6516       \hss\hb@xt@\displaywidth{\#1\glet\bb@upset\@currentlabel}\hss}}%
6517   \else
6518     \eqno\hbox{#1}\glet\bb@upset\@currentlabel}%
6519   \fi
6520   \bb@exp{\def\\\@currentlabel{\[bb@upset]}}
6521 \AtBeginDocument{%
6522   \ifx\bb@noamsmath\relax\else
6523     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6524       \AddToHook{env/equation/begin}{%
6525         \ifnum\bb@thetextdir>\z@
6526           \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6527           \let\@eqnnum\bb@eqnum
6528           \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6529           \chardef\bb@thetextdir\z@
6530           \bb@add\normalfont{\bb@eqnodir}%
6531           \ifcase\bb@eqnpos
6532             \let\bb@puteqno\bb@eqno@flip
6533           \or
6534             \let\bb@puteqno\bb@leqno@flip
6535           \fi
6536           \fi}%
6537         \ifnum\bb@eqnpos=\tw@\else
6538           \def\endequation{\bb@puteqno{\@eqnnum}$$\@ignoretrue}%
6539         \fi
6540         \AddToHook{env/eqnarray/begin}{%
6541           \ifnum\bb@thetextdir>\z@
6542             \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6543             \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6544             \chardef\bb@thetextdir\z@
6545             \bb@add\normalfont{\bb@eqnodir}%
6546             \ifnum\bb@eqnpos=\@ne
6547               \def\@eqnnum{%
6548                 \setbox\z@\hbox{\bb@eqnum}%
6549                 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6550             \else
6551               \let\@eqnnum\bb@eqnum
6552             \fi

```



```

6553     \fi}
6554     % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6555     \expandafter\bb@l@sreplace\csname \endcsname{${\eqno\kern.001pt$}}%
6556 \else % amstex
6557     \bb@l@exp{% Hack to hide maybe undefined conditionals:
6558         \chardef\bb@l@eqnpos=0%
6559         \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6560     \ifnum\bb@l@eqnpos=\@ne
6561         \let\bb@l@ams@lap\hbox
6562     \else
6563         \let\bb@l@ams@lap\llap
6564     \fi
6565     \ExplSyntaxOn % Required by \bb@l@sreplace with \intertext@
6566     \bb@l@sreplace\intertext@\normalbaselines%
6567         {\normalbaselines
6568         \ifx\bb@l@eqnodir\relax\else\bb@l@p@dir\@ne\bb@l@eqnodir\fi}%
6569     \ExplSyntaxOff
6570     \def\bb@l@ams@tagbox#1#2#1{\bb@l@eqnodir#2}% #1=hbox|@lap|flip
6571     \ifx\bb@l@ams@lap\hbox % leqno
6572         \def\bb@l@ams@flip#1{%
6573             \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6574     \else % eqno
6575         \def\bb@l@ams@flip#1{%
6576             \hbox to 0.01pt{\hbox to\displaywidth{\hss#1}\hss}}%
6577     \fi
6578     \def\bb@l@ams@preset#1{%
6579         \def\bb@l@mathboxdir{\def\bb@l@insidemath{1}}%
6580         \ifnum\bb@l@thetextdir>\z@
6581             \edef\bb@l@eqnodir{\noexpand\bb@l@textdir{\the\bb@l@thetextdir}}%
6582             \bb@l@sreplace\textdef@{\hbox}{\bb@l@ams@tagbox\hbox}%
6583             \bb@l@sreplace\maketag@@@{\hbox}{\bb@l@ams@tagbox#1}%
6584         \fi}%
6585     \ifnum\bb@l@eqnpos=\tw@\else
6586         \def\bb@l@ams@equation{%
6587             \def\bb@l@mathboxdir{\def\bb@l@insidemath{1}}%
6588             \ifnum\bb@l@thetextdir>\z@
6589                 \edef\bb@l@eqnodir{\noexpand\bb@l@textdir{\the\bb@l@thetextdir}}%
6590                 \chardef\bb@l@thetextdir\z@
6591                 \bb@l@add\normalfont{\bb@l@eqnodir}%
6592                 \ifcase\bb@l@eqnpos
6593                     \def\veqno##1##2{\bb@l@eqno@flip{##1##2}}%
6594                 \or
6595                     \def\veqno##1##2{\bb@l@leqno@flip{##1##2}}%
6596                 \fi
6597             \fi}%
6598     \AddToHook{env/equation/begin}{\bb@l@ams@equation}%
6599     \AddToHook{env/equation*/begin}{\bb@l@ams@equation}%
6600     \fi
6601     \AddToHook{env/cases/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6602     \AddToHook{env/multline/begin}{\bb@l@ams@preset\hbox}%
6603     \AddToHook{env/gather/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6604     \AddToHook{env/gather*/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6605     \AddToHook{env/align/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6606     \AddToHook{env/align*/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6607     \AddToHook{env/alignat/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6608     \AddToHook{env/alignat*/begin}{\bb@l@ams@preset\bb@l@ams@lap}%
6609     \AddToHook{env/eqnalign/begin}{\bb@l@ams@preset\hbox}%
6610     % Hackish, for proper alignment. Don't ask me why it works!:
6611     \bb@l@exp{% Avoid a 'visible' conditional
6612         \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6613         \\\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6614     \AddToHook{env/flalign/begin}{\bb@l@ams@preset\hbox}%
6615     \AddToHook{env/split/before}{%

```

```

6616     \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6617     \ifnum\bb@thetextdir>\z@
6618         \bb@ifsamestring\@currentvir{equation}%
6619         {\ifx\bb@ams@lap\hbox % leqno
6620             \def\bb@ams@flip#1{%
6621                 \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6622         \else
6623             \def\bb@ams@flip#1{%
6624                 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss#1}}}%
6625         \fi}%
6626     {}%
6627     \fi}%
6628     \fi\fi}
6629 \fi
6630 \def\bb@provide@extra#1{%
6631     % == onchar ==
6632     \ifx\bb@KVP@onchar\@nnil\else
6633         \bb@luaahyphenate
6634         \bb@exp{%
6635             \\\AddToHook{env/document/before}{\select@language{#1}}}%
6636         \directlua{
6637             if Babel.locale_mapped == nil then
6638                 Babel.locale_mapped = true
6639                 Babel.linebreaking.add_before(Babel.locale_map, 1)
6640                 Babel.loc_to_scr = {}
6641                 Babel.chr_to_loc = Babel.chr_to_loc or {}
6642             end
6643             Babel.locale_props[\the\localeid].letters = false
6644         }%
6645         \bb@xin@{ letters }{ \bb@KVP@onchar\space}%
6646         \fin@
6647         \directlua{
6648             Babel.locale_props[\the\localeid].letters = true
6649         }%
6650         \fi
6651         \bb@xin@{ ids }{ \bb@KVP@onchar\space}%
6652         \fin@
6653         \ifx\bb@starthyphens\@undefined % Needed if no explicit selection
6654             \AddBabelHook{babel-onchar}{beforestart}{\bb@starthyphens}%
6655         \fi
6656         \bb@exp{\bb@add\bb@starthyphens
6657             {\bb@patterns@lua{\languagename}}}%
6658         %^^A add error/warning if no script
6659         \directlua{
6660             if Babel.script_blocks['\bb@cl{sbc}'] then
6661                 Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bb@cl{sbc}']
6662                 Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
6663             end
6664         }%
6665         \fi
6666         \bb@xin@{ fonts }{ \bb@KVP@onchar\space}%
6667         \fin@
6668         \bb@ifunset{bb@lsys@\languagename}{\bb@provide@lsys{\languagename}}{%
6669         \bb@ifunset{bb@wdir@\languagename}{\bb@provide@dirs{\languagename}}{%
6670         \directlua{
6671             if Babel.script_blocks['\bb@cl{sbc}'] then
6672                 Babel.loc_to_scr[\the\localeid] =
6673                 Babel.script_blocks['\bb@cl{sbc}']
6674             end}%
6675         \ifx\bb@mapselect\@undefined % TODO. almost the same as mapfont
6676         \AtBeginDocument{%
6677             \bb@patchfont{\bb@mapselect}}%
6678             {\selectfont}}%

```

```

6679     \def\bb@mapselect{%
6680         \let\bb@mapselect\relax
6681         \edef\bb@prefontid{\fontid\font}}%
6682     \def\bb@mapdir##1{%
6683         \begingroup
6684             \setbox\z@\hbox{% Force text mode
6685                 \def\languagename{##1}%
6686                 \let\bb@ifrestoring\@firstoftwo % To avoid font warning
6687                 \bb@switchfont
6688                 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6689                     \directlua{
6690                         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6691                         ['\bb@prefontid'] = \fontid\font\space}%
6692                 \fi}%
6693         \endgroup}%
6694     \fi
6695     \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\languagename}}}%
6696 \fi
6697 % TODO - catch non-valid values
6698 \fi
6699 % == mapfont ==
6700 % For bidi texts, to switch the font based on direction. Old.
6701 \ifx\bb@KVP@mapfont\@nnil\else
6702     \bb@ifsamestring{\bb@KVP@mapfont}{direction}}{%
6703         {\bb@error{unknown-mapfont}}{}}}%
6704     \bb@ifunset{bb@lsys@\languagename}{\bb@provide@lsys{\languagename}}}%
6705     \bb@ifunset{bb@wdir@\languagename}{\bb@provide@dirs{\languagename}}}%
6706     \ifx\bb@mapselect\@undefined % TODO. See onchar.
6707         \AtBeginDocument{%
6708             \bb@patchfont{\bb@mapselect}}%
6709             {\selectfont}}%
6710         \def\bb@mapselect{%
6711             \let\bb@mapselect\relax
6712             \edef\bb@prefontid{\fontid\font}}%
6713         \def\bb@mapdir##1{%
6714             {\def\languagename{##1}%
6715                 \let\bb@ifrestoring\@firstoftwo % avoid font warning
6716                 \bb@switchfont
6717                 \directlua{Babel.fontmap
6718                     [\the\csname bbl@wdir@##1\endcsname]%
6719                     [\bb@prefontid]=\fontid\font}}}%
6720     \fi
6721     \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\languagename}}}%
6722 \fi
6723 % == Line breaking: CJK quotes ==
6724 \ifcase\bb@engine\or
6725     \bb@xin@{/c}{\bb@cl{\lnbrk}}}%
6726     \ifin@
6727         \bb@ifunset{bb@quote@\languagename}}{%
6728             {\directlua{
6729                 Babel.locale_props[\the\localeid].cjk_quotes = {}
6730                 local cs = 'op'
6731                 for c in string.utfvalues(%
6732                     [[\csname bbl@quote@\languagename\endcsname]]) do
6733                     if Babel.cjk_characters[c].c == 'qu' then
6734                         Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6735                     end
6736                     cs = ( cs == 'op') and 'cl' or 'op'
6737                 end
6738             }}%
6739     \fi
6740 \fi
6741 % == Counters: mapdigits ==

```

```

6742 % Native digits
6743 \ifx\bbk@KVP@mapdigits\@nnil\else
6744   \bbk@ifunset{bbk@dgnat@languagename}{}%
6745   {\RequirePackage{luatexbase}%
6746    \bbk@activate@preotf
6747    \directlua{
6748      Babel.digits_mapped = true
6749      Babel.digits = Babel.digits or {}
6750      Babel.digits[\the\localeid] =
6751        table.pack(string.utfvalue('\bbk@cl{dgnat}'))
6752      if not Babel.numbers then
6753        function Babel.numbers(head)
6754          local LOCALE = Babel.attr_locale
6755          local GLYPH = node.id'glyph'
6756          local inmath = false
6757          for item in node.traverse(head) do
6758            if not inmath and item.id == GLYPH then
6759              local temp = node.get_attribute(item, LOCALE)
6760              if Babel.digits[temp] then
6761                local chr = item.char
6762                if chr > 47 and chr < 58 then
6763                  item.char = Babel.digits[temp][chr-47]
6764                end
6765              end
6766            elseif item.id == node.id'math' then
6767              inmath = (item.subtype == 0)
6768            end
6769          end
6770          return head
6771        end
6772      end
6773    }%
6774  \fi
6775 % == transforms ==
6776 \ifx\bbk@KVP@transforms\@nnil\else
6777   \def\bbk@elt##1##2##3{%
6778     \in@{${transforms.}{##1}%
6779     \ifin@
6780     \def\bbk@tempa{##1}%
6781     \bbk@replace\bbk@tempa{transforms.}{}%
6782     \bbk@carg\bbk@transforms{babel\bbk@tempa}{##2}{##3}%
6783   \fi}%
6784 \bbk@exp{%
6785   \\bbk@ifblank{\bbk@cl{dgnat}}%
6786   {\let\\bbk@tempa\relax}%
6787   {\def\\bbk@tempa{%
6788     \\bbk@elt{transforms.prehyphenation}%
6789     {digits.native.1.0}{([0-9])}%
6790     \\bbk@elt{transforms.prehyphenation}%
6791     {digits.native.1.1}{string={\string|0123456789\string|\bbk@cl{dgnat}}}}}%
6792 \ifx\bbk@tempa\relax\else
6793   \toks@\expandafter\expandafter\expandafter{%
6794     \csname bbl@inidata@\languagename\endcsname}%
6795   \bbk@carg\edef{inidata@\languagename}{%
6796     \unexpanded\expandafter{\bbk@tempa}%
6797     \the\toks@}%
6798 \fi
6799 \csname bbl@inidata@\languagename\endcsname
6800 \bbk@release@transforms\relax % \relax closes the last item.
6801 \fi}

```

Start tabular here:

```
6802 \def\localerestoredirs{%
```

```

6803 \ifcase\bb@thetextdir
6804   \ifnum\textdirection=\z@\else\textdir TLT\fi
6805 \else
6806   \ifnum\textdirection=\@ne\else\textdir TRT\fi
6807 \fi
6808 \ifcase\bb@thepardir
6809   \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6810 \else
6811   \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6812 \fi}
6813 \IfBabelLayout{tabular}%
6814   {\chardef\bb@tabular@mode\tw@}% All RTL
6815   {\IfBabelLayout{notabular}%
6816     {\chardef\bb@tabular@mode\z@}%
6817     {\chardef\bb@tabular@mode\@ne}}% Mixed, with LTR cols
6818 \ifnum\bb@bidimode>\@ne % Any lua bidi= except default=1
6819 % Redefine: vrules mess up dirs. TODO: why?
6820 \def\@arstrut{\relax\copy\@arstrutbox}%
6821 \ifcase\bb@tabular@mode\or % 1 = Mixed - default
6822   \let\bb@parabefore\relax
6823   \AddToHook{para/before}{\bb@parabefore}
6824   \AtBeginDocument{%
6825     \bb@replace\@tabular{$}{$}%
6826     \def\bb@insidemath{0}%
6827     \def\bb@parabefore{\localerestoredirs}}%
6828   \ifnum\bb@tabular@mode=\@ne
6829     \bb@ifunset{@tabclassz}{}%
6830     \bb@exp{% Hide conditionals
6831       \\bb@sreplace\\@tabclassz
6832       {\<ifcase>\\@chnum}%
6833       {\\localerestoredirs\<ifcase>\\@chnum}}}%
6834     \@ifpackageloaded{colortbl}%
6835     {\bb@sreplace\@classz
6836       {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6837     {\@ifpackageloaded{array}%
6838       {\bb@exp{% Hide conditionals
6839         \\bb@sreplace\\@classz
6840         {\<ifcase>\\@chnum}%
6841         {\bgroup\\localerestoredirs\<ifcase>\\@chnum}%
6842         \\bb@sreplace\\@classz
6843         {\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6844       {}}%
6845     \fi}%
6846 \or % 2 = All RTL - tabular
6847   \let\bb@parabefore\relax
6848   \AddToHook{para/before}{\bb@parabefore}
6849   \AtBeginDocument{%
6850     \@ifpackageloaded{colortbl}%
6851     {\bb@replace\@tabular{$}{$}%
6852     \def\bb@insidemath{0}%
6853     \def\bb@parabefore{\localerestoredirs}}%
6854     \bb@sreplace\@classz
6855     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6856     {}}%
6857 \fi

```

Very likely the `\output` routine must be patched in a quite general way to make sure the `\bodydir` is set to `\pagedir`. Note outside `\output` they can be different (and often are). For the moment, two *ad hoc* changes.

```

6858 \AtBeginDocument{%
6859   \@ifpackageloaded{multicol}%
6860   {\toks@expandafter{\multi@column@out}%
6861   \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%

```

```

6862     {}}%
6863     \@ifpackageloaded{paracol}%
6864     {\edef\pcol@output{%
6865       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6866     {}}%
6867 \fi
6868 \ifx\bbbl@opt@layout\@nnil\endinput\fi % if no layout

    OMEGA provided a companion to \mathdir (\nextfakemath) for those cases where we did not want
    it to be applied, so that the writing direction of the main text was left unchanged. \bbbl@nextfake
    is an attempt to emulate it, because luatex has removed it without an alternative. Also, \hangindent
    does not honour direction changes by default, so we need to redefine \@hangfrom.

6869 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6870   \def\bbbl@nextfake#1{% non-local changes, use always inside a group!
6871     \bbbl@exp{%
6872       \mathdir\the\bodydir
6873       #1%           Once entered in math, set boxes to restore values
6874       \def\bbbl@insidemath{0}%
6875       \<ifmmode>%
6876         \everyvbox{%
6877           \the\everyvbox
6878           \bodydir\the\bodydir
6879           \mathdir\the\mathdir
6880           \everyhbox{\the\everyhbox}%
6881           \everyvbox{\the\everyvbox}}%
6882         \everyhbox{%
6883           \the\everyhbox
6884           \bodydir\the\bodydir
6885           \mathdir\the\mathdir
6886           \everyhbox{\the\everyhbox}%
6887           \everyvbox{\the\everyvbox}}%
6888         \<fi>}}%
6889   \def\@hangfrom#1{%
6890     \setbox\@tempboxa\hbox{#1}}%
6891     \hangindent\wd\@tempboxa
6892     \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
6893       \shapemode\@ne
6894     \fi
6895     \noindent\box\@tempboxa}
6896 \fi

6897 \IfBabelLayout{tabular}
6898   {\let\bbbl@OL@tabular\@tabular
6899     \bbbl@replace\@tabular{\$}{\bbbl@nextfake$}%
6900     \let\bbbl@NL@tabular\@tabular
6901     \AtBeginDocument{%
6902       \ifx\bbbl@NL@tabular\@tabular\else
6903         \bbbl@exp{\in{\bbbl@nextfake}{\@tabular}}}%
6904       \ifin\@else
6905         \bbbl@replace\@tabular{\$}{\bbbl@nextfake$}%
6906       \fi
6907       \let\bbbl@NL@tabular\@tabular
6908     \fi}}
6909   {}

6910 \IfBabelLayout{lists}
6911   {\let\bbbl@OL@list\list
6912     \bbbl@sreplace\list{\parshape}{\bbbl@listparshape}%
6913     \let\bbbl@NL@list\list
6914     \def\bbbl@listparshape#1#2#3{%
6915       \parshape #1 #2 #3 %
6916       \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
6917         \shapemode\tw@
6918       \fi}}
6919   {}

```

```

6920 \IfBabelLayout{graphics}
6921 {\let\bbl@pictresetdir\relax
6922 \def\bbl@pictsetdir#1{%
6923 \ifcase\bbl@thetextdir
6924 \let\bbl@pictresetdir\relax
6925 \else
6926 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6927 \or\textdir TLT
6928 \else\bodydir TLT \textdir TLT
6929 \fi
6930 % \(\text|par)dir required in pgf:
6931 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6932 \fi}%
6933 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6934 \directlua{
6935 Babel.get_picture_dir = true
6936 Babel.picture_has_bidi = 0
6937 %
6938 function Babel.picture_dir (head)
6939 if not Babel.get_picture_dir then return head end
6940 if Babel.hlist_has_bidi(head) then
6941 Babel.picture_has_bidi = 1
6942 end
6943 return head
6944 end
6945 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6946 "Babel.picture_dir")
6947 }%
6948 \AtBeginDocument{%
6949 \def\LS@rot{%
6950 \setbox\@outputbox\vbox{%
6951 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6952 \long\def\put(#1,#2)#3{%
6953 \@killglue
6954 % Try:
6955 \ifx\bbl@pictresetdir\relax
6956 \def\bbl@tempc{0}%
6957 \else
6958 \directlua{
6959 Babel.get_picture_dir = true
6960 Babel.picture_has_bidi = 0
6961 }%
6962 \setbox\z@\hb@xt@z@{%
6963 \@defaultunitsset\@tempdimc{#1}\unitlength
6964 \kern\@tempdimc
6965 #3\hss}% TODO: #3 executed twice (below). That's bad.
6966 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6967 \fi
6968 % Do:
6969 \@defaultunitsset\@tempdimc{#2}\unitlength
6970 \raise\@tempdimc\hb@xt@z@{%
6971 \@defaultunitsset\@tempdimc{#1}\unitlength
6972 \kern\@tempdimc
6973 {\ifnum\bbl@tempc>z@\bbl@pictresetdir\fi#3}\hss}%
6974 \ignorespaces}%
6975 \MakeRobust\put}%
6976 \AtBeginDocument
6977 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir@gobble}%
6978 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6979 \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6980 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6981 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6982 \fi

```

```

6983 \ifx\tikzpicture\undefined\else
6984 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6985 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6986 \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6987 \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6988 \fi
6989 \ifx\tcolorbox\undefined\else
6990 \def\tcb@drawing@env@begin{%
6991 \csname tcb@before@\tcb@split@state\endcsname
6992 \bbl@pictsetdir\tw@
6993 \begin{\kvtcb@graphenv}%
6994 \tcb@bbdraw
6995 \tcb@apply@graph@patches}%
6996 \def\tcb@drawing@env@end{%
6997 \end{\kvtcb@graphenv}%
6998 \bbl@pictresetdir
6999 \csname tcb@after@\tcb@split@state\endcsname}%
7000 \fi
7001 }}
7002 {}

```

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`.

```

7003 \IfBabelLayout{counters*}%
7004 {\bbl@add\bbl@opt@layout{.counters.}%
7005 \directlua{
7006 \luaexec{
7007 \luaexec{
7008 }{}
7009 \IfBabelLayout{counters}%
7010 {\let\bbl@0L@textsuperscript\textsuperscript
7011 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7012 \let\bbl@latinarabic=\@arabic
7013 \let\bbl@0L@arabic\@arabic
7014 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7015 \ifpackagewith{babel}{bidi=default}%
7016 {\let\bbl@asciroman=\@roman
7017 \let\bbl@0L@roman\@roman
7018 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}
7019 \let\bbl@asciRoman=\@Roman
7020 \let\bbl@0L@roman\@Roman
7021 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}
7022 \let\bbl@0L@labelenumii\labelenumii
7023 \def\labelenumii{\theenumii}%
7024 \let\bbl@0L@p@enumiii\p@enumiii
7025 \def\p@enumiii{\p@enumii}\theenumii{}}{}{}
7026 <@Footnote changes@>
7027 \IfBabelLayout{footnotes}%
7028 {\let\bbl@0L@footnote\footnote
7029 \BabelFootnote\footnote\languagename{}}{}%
7030 \BabelFootnote\localfootnote\languagename{}}{}%
7031 \BabelFootnote\mainfootnote{}}{}{}
7032 {}

```

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.

```

7033 \IfBabelLayout{extras}%
7034 {\bbl@ncarg\let\bbl@0L@underline{underline }%
7035 \bbl@carg\bbl@sreplace{underline }%
7036 {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
7037 \bbl@carg\bbl@sreplace{underline }%
7038 {\m@th$}{\m@th$\egroup}%
7039 \let\bbl@0L@LaTeXe\LaTeXe

```



```

7040 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7041 \if b\expandafter\@car\f@series\@nil\boldmath\fi
7042 \babelsublr{%
7043 \LaTeX\kern.15em2\bb@nextfake$_{\textstyle\varepsilon}$}}
7044 {}
7045 </luatex>

```

### 10.13.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.

```

7046 <{*transforms}>
7047 Babel.linebreaking.replacements = {}
7048 Babel.linebreaking.replacements[0] = {} -- pre
7049 Babel.linebreaking.replacements[1] = {} -- post
7050
7051 function Babel.tovalue(v)
7052   if type(v) == 'table' then
7053     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7054   else
7055     return v
7056   end
7057 end
7058
7059 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7060
7061 function Babel.set_hboxed(head, gc)
7062   for item in node.traverse(head) do
7063     node.set_attribute(item, Babel.attr_hboxed, 1)
7064   end
7065   return head
7066 end
7067
7068 Babel.fetch_subtext = {}
7069
7070 Babel.ignore_pre_char = function(node)
7071   return (node.lang == Babel.nohyphenation)
7072 end
7073
7074 -- Merging both functions doesn't seem feasible, because there are too
7075 -- many differences.
7076 Babel.fetch_subtext[0] = function(head)
7077   local word_string = ''
7078   local word_nodes = {}
7079   local lang
7080   local item = head
7081   local inmath = false
7082
7083   while item do
7084
7085     if item.id == 11 then
7086       inmath = (item.subtype == 0)
7087     end
7088

```

```

7089     if inmath then
7090         -- pass
7091
7092     elseif item.id == 29 then
7093         local locale = node.get_attribute(item, Babel.attr_locale)
7094
7095         if lang == locale or lang == nil then
7096             lang = lang or locale
7097             if Babel.ignore_pre_char(item) then
7098                 word_string = word_string .. Babel.us_char
7099             else
7100                 if node.has_attribute(item, Babel.attr_hboxed) then
7101                     word_string = word_string .. Babel.us_char
7102                 else
7103                     word_string = word_string .. unicode.utf8.char(item.char)
7104                 end
7105             end
7106             word_nodes[#word_nodes+1] = item
7107         else
7108             break
7109         end
7110
7111     elseif item.id == 12 and item.subtype == 13 then
7112         if node.has_attribute(item, Babel.attr_hboxed) then
7113             word_string = word_string .. Babel.us_char
7114         else
7115             word_string = word_string .. ' '
7116         end
7117         word_nodes[#word_nodes+1] = item
7118
7119         -- Ignore leading unrecognized nodes, too.
7120     elseif word_string ~= '' then
7121         word_string = word_string .. Babel.us_char
7122         word_nodes[#word_nodes+1] = item -- Will be ignored
7123     end
7124
7125     item = item.next
7126 end
7127
7128 -- Here and above we remove some trailing chars but not the
7129 -- corresponding nodes. But they aren't accessed.
7130 if word_string:sub(-1) == ' ' then
7131     word_string = word_string:sub(1,-2)
7132 end
7133 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7134 return word_string, word_nodes, item, lang
7135 end
7136
7137 Babel.fetch_subtext[1] = function(head)
7138     local word_string = ''
7139     local word_nodes = {}
7140     local lang
7141     local item = head
7142     local inmath = false
7143
7144     while item do
7145
7146         if item.id == 11 then
7147             inmath = (item.subtype == 0)
7148         end
7149
7150         if inmath then
7151             -- pass

```

```

7152
7153     elsif item.id == 29 then
7154         if item.lang == lang or lang == nil then
7155             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7156                 lang = lang or item.lang
7157                 if node.has_attribute(item, Babel.attr_hboxed) then
7158                     word_string = word_string .. Babel.us_char
7159                 else
7160                     word_string = word_string .. unicode.utf8.char(item.char)
7161                 end
7162                 word_nodes[#word_nodes+1] = item
7163             end
7164         else
7165             break
7166         end
7167
7168     elsif item.id == 7 and item.subtype == 2 then
7169         if node.has_attribute(item, Babel.attr_hboxed) then
7170             word_string = word_string .. Babel.us_char
7171         else
7172             word_string = word_string .. '='
7173         end
7174         word_nodes[#word_nodes+1] = item
7175
7176     elsif item.id == 7 and item.subtype == 3 then
7177         if node.has_attribute(item, Babel.attr_hboxed) then
7178             word_string = word_string .. Babel.us_char
7179         else
7180             word_string = word_string .. '|'
7181         end
7182         word_nodes[#word_nodes+1] = item
7183
7184         -- (1) Go to next word if nothing was found, and (2) implicitly
7185         -- remove leading USs.
7186         elsif word_string == '' then
7187             -- pass
7188
7189         -- This is the responsible for splitting by words.
7190         elsif (item.id == 12 and item.subtype == 13) then
7191             break
7192
7193     else
7194         word_string = word_string .. Babel.us_char
7195         word_nodes[#word_nodes+1] = item -- Will be ignored
7196     end
7197
7198     item = item.next
7199 end
7200
7201 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7202 return word_string, word_nodes, item, lang
7203 end
7204
7205 function Babel.pre_hyphenate_replace(head)
7206     Babel.hyphenate_replace(head, 0)
7207 end
7208
7209 function Babel.post_hyphenate_replace(head)
7210     Babel.hyphenate_replace(head, 1)
7211 end
7212
7213 Babel.us_char = string.char(31)
7214

```

```

7215 function Babel.hyphenate_replace(head, mode)
7216   local u = unicode.utf8
7217   local lbkr = Babel.linebreaking.replacements[mode]
7218   local tovalue = Babel.tovalue
7219
7220   local word_head = head
7221
7222   while true do -- for each subtext block
7223
7224     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7225
7226     if Babel.debug then
7227       print()
7228       print((mode == 0) and '@@@@<' or '@@@@>', w)
7229     end
7230
7231     if nw == nil and w == '' then break end
7232
7233     if not lang then goto next end
7234     if not lbkr[lang] then goto next end
7235
7236     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7237     -- loops are nested.
7238     for k=1, #lbkr[lang] do
7239       local p = lbkr[lang][k].pattern
7240       local r = lbkr[lang][k].replace
7241       local attr = lbkr[lang][k].attr or -1
7242
7243       if Babel.debug then
7244         print('*****', p, mode)
7245       end
7246
7247       -- This variable is set in some cases below to the first *byte*
7248       -- after the match, either as found by u.match (faster) or the
7249       -- computed position based on sc if w has changed.
7250       local last_match = 0
7251       local step = 0
7252
7253       -- For every match.
7254       while true do
7255         if Babel.debug then
7256           print('=====' )
7257         end
7258         local new -- used when inserting and removing nodes
7259         local dummy_node -- used by after
7260
7261         local matches = { u.match(w, p, last_match) }
7262
7263         if #matches < 2 then break end
7264
7265         -- Get and remove empty captures (with ()'s, which return a
7266         -- number with the position), and keep actual captures
7267         -- (from (...)), if any, in matches.
7268         local first = table.remove(matches, 1)
7269         local last = table.remove(matches, #matches)
7270         -- Non re-fetched substrings may contain \31, which separates
7271         -- subsubstrings.
7272         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7273
7274         local save_last = last -- with A()BC()D, points to D
7275
7276         -- Fix offsets, from bytes to unicode. Explained above.
7277         first = u.len(w:sub(1, first-1)) + 1

```

```

7278     last = u.len(w:sub(1, last-1)) -- now last points to C
7279
7280     -- This loop stores in a small table the nodes
7281     -- corresponding to the pattern. Used by 'data' to provide a
7282     -- predictable behavior with 'insert' (w_nodes is modified on
7283     -- the fly), and also access to 'remove'd nodes.
7284     local sc = first-1          -- Used below, too
7285     local data_nodes = {}
7286
7287     local enabled = true
7288     for q = 1, last-first+1 do
7289         data_nodes[q] = w_nodes[sc+q]
7290         if enabled
7291             and attr > -1
7292             and not node.has_attribute(data_nodes[q], attr)
7293         then
7294             enabled = false
7295         end
7296     end
7297
7298     -- This loop traverses the matched substring and takes the
7299     -- corresponding action stored in the replacement list.
7300     -- sc = the position in substr nodes / string
7301     -- rc = the replacement table index
7302     local rc = 0
7303
7304     ----- TODO. dummy_node?
7305     while rc < last-first+1 or dummy_node do -- for each replacement
7306         if Babel.debug then
7307             print('.....', rc + 1)
7308         end
7309         sc = sc + 1
7310         rc = rc + 1
7311
7312         if Babel.debug then
7313             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7314             local ss = ''
7315             for itt in node.traverse(head) do
7316                 if itt.id == 29 then
7317                     ss = ss .. unicode.utf8.char(itt.char)
7318                 else
7319                     ss = ss .. '{' .. itt.id .. '}'
7320                 end
7321             end
7322             print('*****', ss)
7323         end
7324
7325         local crep = r[rc]
7326         local item = w_nodes[sc]
7327         local item_base = item
7328         local placeholder = Babel.us_char
7329         local d
7330
7331         if crep and crep.data then
7332             item_base = data_nodes[crep.data]
7333         end
7334
7335         if crep then
7336             step = crep.step or step
7337         end
7338
7339         if crep and crep.after then

```

```

7341     crep.insert = true
7342     if dummy_node then
7343         item = dummy_node
7344     else -- TODO. if there is a node after?
7345         d = node.copy(item_base)
7346         head, item = node.insert_after(head, item, d)
7347         dummy_node = item
7348     end
7349 end
7350
7351 if crep and not crep.after and dummy_node then
7352     node.remove(head, dummy_node)
7353     dummy_node = nil
7354 end
7355
7356 if not enabled then
7357     last_match = save_last
7358     goto next
7359
7360 elseif crep and next(crep) == nil then -- = {}
7361     if step == 0 then
7362         last_match = save_last -- Optimization
7363     else
7364         last_match = utf8.offset(w, sc+step)
7365     end
7366     goto next
7367
7368 elseif crep == nil or crep.remove then
7369     node.remove(head, item)
7370     table.remove(w_nodes, sc)
7371     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7372     sc = sc - 1 -- Nothing has been inserted.
7373     last_match = utf8.offset(w, sc+1+step)
7374     goto next
7375
7376 elseif crep and crep.kashida then -- Experimental
7377     node.set_attribute(item,
7378         Babel.attr_kashida,
7379         crep.kashida)
7380     last_match = utf8.offset(w, sc+1+step)
7381     goto next
7382
7383 elseif crep and crep.string then
7384     local str = crep.string(matches)
7385     if str == '' then -- Gather with nil
7386         node.remove(head, item)
7387         table.remove(w_nodes, sc)
7388         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7389         sc = sc - 1 -- Nothing has been inserted.
7390     else
7391         local loop_first = true
7392         for s in string.utfvalues(str) do
7393             d = node.copy(item_base)
7394             d.char = s
7395             if loop_first then
7396                 loop_first = false
7397                 head, new = node.insert_before(head, item, d)
7398                 if sc == 1 then
7399                     word_head = head
7400                 end
7401                 w_nodes[sc] = d
7402                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7403             else

```

```

7404         sc = sc + 1
7405         head, new = node.insert_before(head, item, d)
7406         table.insert(w_nodes, sc, new)
7407         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7408     end
7409     if Babel.debug then
7410         print('.....', 'str')
7411         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7412     end
7413     end -- for
7414     node.remove(head, item)
7415 end -- if ''
7416 last_match = utf8.offset(w, sc+1+step)
7417 goto next
7418
7419 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7420     d = node.new(7, 3) -- (disc, regular)
7421     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7422     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7423     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7424     d.attr = item_base.attr
7425     if crep.pre == nil then -- TeXbook p96
7426         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7427     else
7428         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7429     end
7430     placeholder = '|'
7431     head, new = node.insert_before(head, item, d)
7432
7433 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7434     -- ERROR
7435
7436 elseif crep and crep.penalty then
7437     d = node.new(14, 0) -- (penalty, userpenalty)
7438     d.attr = item_base.attr
7439     d.penalty = tovalue(crep.penalty)
7440     head, new = node.insert_before(head, item, d)
7441
7442 elseif crep and crep.space then
7443     -- 655360 = 10 pt = 10 * 65536 sp
7444     d = node.new(12, 13) -- (glue, spaceskip)
7445     local quad = font.getfont(item_base.font).size or 655360
7446     node.setglue(d, tovalue(crep.space[1]) * quad,
7447                 tovalue(crep.space[2]) * quad,
7448                 tovalue(crep.space[3]) * quad)
7449     if mode == 0 then
7450         placeholder = ' '
7451     end
7452     head, new = node.insert_before(head, item, d)
7453
7454 elseif crep and crep.norule then
7455     -- 655360 = 10 pt = 10 * 65536 sp
7456     d = node.new(2, 3) -- (rule, empty) = \no*rule
7457     local quad = font.getfont(item_base.font).size or 655360
7458     d.width = tovalue(crep.norule[1]) * quad
7459     d.height = tovalue(crep.norule[2]) * quad
7460     d.depth = tovalue(crep.norule[3]) * quad
7461     head, new = node.insert_before(head, item, d)
7462
7463 elseif crep and crep.spacefactor then
7464     d = node.new(12, 13) -- (glue, spaceskip)
7465     local base_font = font.getfont(item_base.font)
7466     node.setglue(d,

```

```

7467         tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7468         tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7469         tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7470     if mode == 0 then
7471         placeholder = ' '
7472     end
7473     head, new = node.insert_before(head, item, d)
7474
7475 elseif mode == 0 and crep and crep.space then
7476     -- ERROR
7477
7478 elseif crep and crep.kern then
7479     d = node.new(13, 1)      -- (kern, user)
7480     local quad = font.getfont(item_base.font).size or 655360
7481     d.attr = item_base.attr
7482     d.kern = tovalue(crep.kern) * quad
7483     head, new = node.insert_before(head, item, d)
7484
7485 elseif crep and crep.node then
7486     d = node.new(crep.node[1], crep.node[2])
7487     d.attr = item_base.attr
7488     head, new = node.insert_before(head, item, d)
7489
7490 end -- i.e., replacement cases
7491
7492 -- Shared by disc, space(factor), kern, node and penalty.
7493 if sc == 1 then
7494     word_head = head
7495 end
7496 if crep.insert then
7497     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7498     table.insert(w_nodes, sc, new)
7499     last = last + 1
7500 else
7501     w_nodes[sc] = d
7502     node.remove(head, item)
7503     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7504 end
7505
7506 last_match = utf8.offset(w, sc+1+step)
7507
7508 ::next::
7509
7510 end -- for each replacement
7511
7512 if Babel.debug then
7513     print('.....', '/')
7514     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7515 end
7516
7517 if dummy_node then
7518     node.remove(head, dummy_node)
7519     dummy_node = nil
7520 end
7521
7522 end -- for match
7523
7524 end -- for patterns
7525
7526 ::next::
7527 word_head = nw
7528 end -- for substring
7529 return head

```



```

7530 end
7531
7532 -- This table stores capture maps, numbered consecutively
7533 Babel.capture_maps = {}
7534
7535 -- The following functions belong to the next macro
7536 function Babel.capture_func(key, cap)
7537   local ret = "[" .. cap:gsub('{{[0-9]}}', "")..m[%1]..["] .. "]"
7538   local cnt
7539   local u = unicode.utf8
7540   ret, cnt = ret:gsub('{{[0-9]}|(^|+)|(.-)}', Babel.capture_func_map)
7541   if cnt == 0 then
7542     ret = u.gsub(ret, '{{%x%x%x%x+}}',
7543       function (n)
7544         return u.char(tonumber(n, 16))
7545       end)
7546   end
7547   ret = ret:gsub("%[%[%]%.%", '')
7548   ret = ret:gsub("%.%[%[%]%", '')
7549   return key .. "[=function(m) return ] .. ret .. [[ end]]
7550 end
7551
7552 function Babel.capt_map(from, mapno)
7553   return Babel.capture_maps[mapno][from] or from
7554 end
7555
7556 -- Handle the {n|abc|ABC} syntax in captures
7557 function Babel.capture_func_map(capno, from, to)
7558   local u = unicode.utf8
7559   from = u.gsub(from, '{{%x%x%x%x+}}',
7560     function (n)
7561       return u.char(tonumber(n, 16))
7562     end)
7563   to = u.gsub(to, '{{%x%x%x%x+}}',
7564     function (n)
7565       return u.char(tonumber(n, 16))
7566     end)
7567   local froms = {}
7568   for s in string.utfcharacters(from) do
7569     table.insert(froms, s)
7570   end
7571   local cnt = 1
7572   table.insert(Babel.capture_maps, {})
7573   local mlen = table.getn(Babel.capture_maps)
7574   for s in string.utfcharacters(to) do
7575     Babel.capture_maps[mlen][froms[cnt]] = s
7576     cnt = cnt + 1
7577   end
7578   return "]"..Babel.capt_map(m[" .. capno .. "], " ..
7579     (mlen) .. " ) .. " .. "[["
7580 end
7581
7582 -- Create/Extend reversed sorted list of kashida weights:
7583 function Babel.capture_kashida(key, wt)
7584   wt = tonumber(wt)
7585   if Babel.kashida_wts then
7586     for p, q in ipairs(Babel.kashida_wts) do
7587       if wt == q then
7588         break
7589       elseif wt > q then
7590         table.insert(Babel.kashida_wts, p, wt)
7591         break
7592       elseif table.getn(Babel.kashida_wts) == p then

```

```

7593         table.insert(Babel.kashida_wts, wt)
7594     end
7595 end
7596 else
7597     Babel.kashida_wts = { wt }
7598 end
7599 return 'kashida = ' .. wt
7600 end
7601
7602 function Babel.capture_node(id, subtype)
7603     local sbt = 0
7604     for k, v in pairs(node.subtypes(id)) do
7605         if v == subtype then sbt = k end
7606     end
7607     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7608 end
7609
7610 -- Experimental: applies prehyphenation transforms to a string (letters
7611 -- and spaces).
7612 function Babel.string_prehyphenation(str, locale)
7613     local n, head, last, res
7614     head = node.new(8, 0) -- dummy (hack just to start)
7615     last = head
7616     for s in string.utfvalues(str) do
7617         if s == 20 then
7618             n = node.new(12, 0)
7619         else
7620             n = node.new(29, 0)
7621             n.char = s
7622         end
7623         node.set_attribute(n, Babel.attr_locale, locale)
7624         last.next = n
7625         last = n
7626     end
7627     head = Babel.hyphenate_replace(head, 0)
7628     res = ''
7629     for n in node.traverse(head) do
7630         if n.id == 12 then
7631             res = res .. ' '
7632         elseif n.id == 29 then
7633             res = res .. unicode.utf8.char(n.char)
7634         end
7635     end
7636     tex.print(res)
7637 end
7638 </transforms>

```

## 10.14 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):

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

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

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

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

```

7639 (*basic-r)
7640 Babel.bidi_enabled = true
7641
7642 require('babel-data-bidi.lua')
7643
7644 local characters = Babel.characters
7645 local ranges = Babel.ranges
7646
7647 local DIR = node.id("dir")
7648
7649 local function dir_mark(head, from, to, outer)
7650   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7651   local d = node.new(DIR)
7652   d.dir = '+' .. dir
7653   node.insert_before(head, from, d)
7654   d = node.new(DIR)
7655   d.dir = '-' .. dir
7656   node.insert_after(head, to, d)
7657 end
7658
7659 function Babel.bidi(head, ispar)
7660   local first_n, last_n           -- first and last char with nums
7661   local last_es                   -- an auxiliary 'last' used with nums
7662   local first_d, last_d           -- first and last char in L/R block
7663   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):

```

7664   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7665   local strong_lr = (strong == 'l') and 'l' or 'r'
7666   local outer = strong
7667
7668   local new_dir = false
7669   local first_dir = false
7670   local inmath = false
7671
7672   local last_lr
7673
7674   local type_n = ''
7675

```

```

7676 for item in node.traverse(head) do
7677
7678   -- three cases: glyph, dir, otherwise
7679   if item.id == node.id'glyph'
7680     or (item.id == 7 and item.subtype == 2) then
7681
7682     local itemchar
7683     if item.id == 7 and item.subtype == 2 then
7684       itemchar = item.replace.char
7685     else
7686       itemchar = item.char
7687     end
7688     local chardata = characters[itemchar]
7689     dir = chardata and chardata.d or nil
7690     if not dir then
7691       for nn, et in ipairs(ranges) do
7692         if itemchar < et[1] then
7693           break
7694         elseif itemchar <= et[2] then
7695           dir = et[3]
7696           break
7697         end
7698       end
7699     end
7700     dir = dir or '\l'
7701     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).

```

7702   if new_dir then
7703     attr_dir = 0
7704     for at in node.traverse(item.attr) do
7705       if at.number == Babel.attr_dir then
7706         attr_dir = at.value & 0x3
7707       end
7708     end
7709     if attr_dir == 1 then
7710       strong = 'r'
7711     elseif attr_dir == 2 then
7712       strong = 'al'
7713     else
7714       strong = '\l'
7715     end
7716     strong_lr = (strong == '\l') and '\l' or 'r'
7717     outer = strong_lr
7718     new_dir = false
7719   end
7720
7721   if dir == 'nsm' then dir = strong end           -- W1

```

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

```

7722   dir_real = dir           -- We need dir_real to set strong below
7723   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:

```

7724   if strong == 'al' then
7725     if dir == 'en' then dir = 'an' end           -- W2
7726     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7727     strong_lr = 'r'                               -- W3
7728   end

```

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

```

7729     elseif item.id == node.id'dir' and not inmath then
7730         new_dir = true
7731         dir = nil
7732     elseif item.id == node.id'math' then
7733         inmath = (item.subtype == 0)
7734     else
7735         dir = nil          -- Not a char
7736     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, i.e., 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>.

```

7737     if dir == 'en' or dir == 'an' or dir == 'et' then
7738         if dir ~= 'et' then
7739             type_n = dir
7740         end
7741         first_n = first_n or item
7742         last_n = last_es or item
7743         last_es = nil
7744     elseif dir == 'es' and last_n then -- W3+W6
7745         last_es = item
7746     elseif dir == 'cs' then          -- it's right - do nothing
7747     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7748         if strong_lr == 'r' and type_n ~= '' then
7749             dir_mark(head, first_n, last_n, 'r')
7750         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7751             dir_mark(head, first_n, last_n, 'r')
7752             dir_mark(head, first_d, last_d, outer)
7753             first_d, last_d = nil, nil
7754         elseif strong_lr == 'l' and type_n ~= '' then
7755             last_d = last_n
7756         end
7757         type_n = ''
7758         first_n, last_n = nil, nil
7759     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:

```

7760     if dir == 'l' or dir == 'r' then
7761         if dir ~= outer then
7762             first_d = first_d or item
7763             last_d = item
7764         elseif first_d and dir ~= strong_lr then
7765             dir_mark(head, first_d, last_d, outer)
7766             first_d, last_d = nil, nil
7767         end
7768     end

```

**Mirroring.** Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last\_lr is nil) of an R text, they are mirrored directly. Numbers in R mode are processed. It should not be done, but it doesn't hurt.

```

7769     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7770         item.char = characters[item.char] and
7771             characters[item.char].m or item.char
7772     elseif (dir or new_dir) and last_lr ~= item then
7773         local mir = outer .. strong_lr .. (dir or outer)

```

```

7774     if mir == 'rrr' or mir == 'lrr' or mir == 'rll' or mir == 'rlr' then
7775         for ch in node.traverse(node.next(last_lr)) do
7776             if ch == item then break end
7777             if ch.id == node.id'glyph' and characters[ch.char] then
7778                 ch.char = characters[ch.char].m or ch.char
7779             end
7780         end
7781     end
7782 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).

```

7783     if dir == 'l' or dir == 'r' then
7784         last_lr = item
7785         strong = dir_real          -- Don't search back - best save now
7786         strong_lr = (strong == 'l') and 'l' or 'r'
7787     elseif new_dir then
7788         last_lr = nil
7789     end
7790 end

```

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

```

7791     if last_lr and outer == 'r' then
7792         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7793             if characters[ch.char] then
7794                 ch.char = characters[ch.char].m or ch.char
7795             end
7796         end
7797     end
7798     if first_n then
7799         dir_mark(head, first_n, last_n, outer)
7800     end
7801     if first_d then
7802         dir_mark(head, first_d, last_d, outer)
7803     end

```

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

```

7804     return node.prev(head) or head
7805 end
7806 </basic-r>

```

And here the Lua code for bidi=basic:

```

7807 <{*basic}
7808 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7809
7810 Babel.fontmap = Babel.fontmap or {}
7811 Babel.fontmap[0] = {}          -- l
7812 Babel.fontmap[1] = {}          -- r
7813 Babel.fontmap[2] = {}          -- al/an
7814
7815 -- To cancel mirroring. Also OML, OMS, U?
7816 Babel.symbol_fonts = Babel.symbol_fonts or {}
7817 Babel.symbol_fonts[font.id('tenln')] = true
7818 Babel.symbol_fonts[font.id('tenlnw')] = true
7819 Babel.symbol_fonts[font.id('tencirc')] = true
7820 Babel.symbol_fonts[font.id('tencircw')] = true
7821
7822 Babel.bidi_enabled = true
7823 Babel.mirroring_enabled = true
7824
7825 require('babel-data-bidi.lua')
7826
7827 local characters = Babel.characters

```

```

7828 local ranges = Babel.ranges
7829
7830 local DIR = node.id('dir')
7831 local GLYPH = node.id('glyph')
7832
7833 local function insert_implicit(head, state, outer)
7834   local new_state = state
7835   if state.sim and state.eim and state.sim ~= state.eim then
7836     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7837     local d = node.new(DIR)
7838     d.dir = '+' .. dir
7839     node.insert_before(head, state.sim, d)
7840     local d = node.new(DIR)
7841     d.dir = '-' .. dir
7842     node.insert_after(head, state.eim, d)
7843   end
7844   new_state.sim, new_state.eim = nil, nil
7845   return head, new_state
7846 end
7847
7848 local function insert_numeric(head, state)
7849   local new
7850   local new_state = state
7851   if state.san and state.ean and state.san ~= state.ean then
7852     local d = node.new(DIR)
7853     d.dir = '+TLT'
7854     _, new = node.insert_before(head, state.san, d)
7855     if state.san == state.sim then state.sim = new end
7856     local d = node.new(DIR)
7857     d.dir = '-TLT'
7858     _, new = node.insert_after(head, state.ean, d)
7859     if state.ean == state.eim then state.eim = new end
7860   end
7861   new_state.san, new_state.ean = nil, nil
7862   return head, new_state
7863 end
7864
7865 local function glyph_not_symbol_font(node)
7866   if node.id == GLYPH then
7867     return not Babel.symbol_fonts[node.font]
7868   else
7869     return false
7870   end
7871 end
7872
7873 -- TODO - \hbox with an explicit dir can lead to wrong results
7874 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7875 -- was made to improve the situation, but the problem is the 3-dir
7876 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7877 -- well.
7878
7879 function Babel.bidi(head, ispar, hdir)
7880   local d -- d is used mainly for computations in a loop
7881   local prev_d = ''
7882   local new_d = false
7883
7884   local nodes = {}
7885   local outer_first = nil
7886   local inmath = false
7887
7888   local glue_d = nil
7889   local glue_i = nil
7890

```

```

7891 local has_en = false
7892 local first_et = nil
7893
7894 local has_hyperlink = false
7895
7896 local ATDIR = Babel.attr_dir
7897 local attr_d, temp
7898 local locale_d
7899
7900 local save_outer
7901 local locale_d = node.get_attribute(head, ATDIR)
7902 if locale_d then
7903     locale_d = locale_d & 0x3
7904     save_outer = (locale_d == 0 and 'l') or
7905                 (locale_d == 1 and 'r') or
7906                 (locale_d == 2 and 'al')
7907 elseif ispar then -- Or error? Shouldn't happen
7908     -- when the callback is called, we are just _after_ the box,
7909     -- and the textdir is that of the surrounding text
7910     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7911 else -- Empty box
7912     save_outer = ('TRT' == hdir) and 'r' or 'l'
7913 end
7914 local outer = save_outer
7915 local last = outer
7916 -- 'al' is only taken into account in the first, current loop
7917 if save_outer == 'al' then save_outer = 'r' end
7918
7919 local fontmap = Babel.fontmap
7920
7921 for item in node.traverse(head) do
7922
7923     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
7924     locale_d = node.get_attribute(item, ATDIR)
7925     node.set_attribute(item, ATDIR, 0x80)
7926
7927     -- In what follows, #node is the last (previous) node, because the
7928     -- current one is not added until we start processing the neutrals.
7929     -- three cases: glyph, dir, otherwise
7930     if glyph_not_symbol_font(item)
7931         or (item.id == 7 and item.subtype == 2) then
7932
7933         if locale_d == 0x80 then goto nextnode end
7934
7935         local d_font = nil
7936         local item_r
7937         if item.id == 7 and item.subtype == 2 then
7938             item_r = item.replace -- automatic discs have just 1 glyph
7939         else
7940             item_r = item
7941         end
7942
7943         local chardata = characters[item_r.char]
7944         d = chardata and chardata.d or nil
7945         if not d or d == 'nsm' then
7946             for nn, et in ipairs(ranges) do
7947                 if item_r.char < et[1] then
7948                     break
7949                 elseif item_r.char <= et[2] then
7950                     if not d then d = et[3]
7951                     elseif d == 'nsm' then d_font = et[3]
7952                     end
7953                     break

```



```

7954         end
7955     end
7956 end
7957 d = d or 'l'
7958
7959 -- A short 'pause' in bidi for mapfont
7960 -- %%%% TODO. move if fontmap here
7961 d_font = d_font or d
7962 d_font = (d_font == 'l' and 0) or
7963         (d_font == 'nsm' and 0) or
7964         (d_font == 'r' and 1) or
7965         (d_font == 'al' and 2) or
7966         (d_font == 'an' and 2) or nil
7967 if d_font and fontmap and fontmap[d_font][item_r.font] then
7968     item_r.font = fontmap[d_font][item_r.font]
7969 end
7970
7971 if new_d then
7972     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7973     if inmath then
7974         attr_d = 0
7975     else
7976         attr_d = locale_d & 0x3
7977     end
7978     if attr_d == 1 then
7979         outer_first = 'r'
7980         last = 'r'
7981     elseif attr_d == 2 then
7982         outer_first = 'r'
7983         last = 'al'
7984     else
7985         outer_first = 'l'
7986         last = 'l'
7987     end
7988     outer = last
7989     has_en = false
7990     first_et = nil
7991     new_d = false
7992 end
7993
7994 if glue_d then
7995     if (d == 'l' and 'l' or 'r') ~= glue_d then
7996         table.insert(nodes, {glue_i, 'on', nil})
7997     end
7998     glue_d = nil
7999     glue_i = nil
8000 end
8001
8002 elseif item.id == DIR then
8003     d = nil
8004     new_d = true
8005
8006 elseif item.id == node.id'glue' and item.subtype == 13 then
8007     glue_d = d
8008     glue_i = item
8009     d = nil
8010
8011 elseif item.id == node.id'math' then
8012     inmath = (item.subtype == 0)
8013
8014 elseif item.id == 8 and item.subtype == 19 then
8015     has_hyperlink = true
8016

```

```

8017 else
8018   d = nil
8019 end
8020
8021 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8022 if last == 'al' and d == 'en' then
8023   d = 'an'             -- W3
8024 elseif last == 'al' and (d == 'et' or d == 'es') then
8025   d = 'on'             -- W6
8026 end
8027
8028 -- EN + CS/ES + EN     -- W4
8029 if d == 'en' and #nodes >= 2 then
8030   if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8031     and nodes[#nodes-1][2] == 'en' then
8032     nodes[#nodes][2] = 'en'
8033   end
8034 end
8035
8036 -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
8037 if d == 'an' and #nodes >= 2 then
8038   if (nodes[#nodes][2] == 'cs')
8039     and nodes[#nodes-1][2] == 'an' then
8040     nodes[#nodes][2] = 'an'
8041   end
8042 end
8043
8044 -- ET/EN               -- W5 + W7->l / W6->on
8045 if d == 'et' then
8046   first_et = first_et or (#nodes + 1)
8047 elseif d == 'en' then
8048   has_en = true
8049   first_et = first_et or (#nodes + 1)
8050 elseif first_et then    -- d may be nil here !
8051   if has_en then
8052     if last == 'l' then
8053       temp = 'l'       -- W7
8054     else
8055       temp = 'en'     -- W5
8056     end
8057   else
8058     temp = 'on'       -- W6
8059   end
8060   for e = first_et, #nodes do
8061     if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8062   end
8063   first_et = nil
8064   has_en = false
8065 end
8066
8067 -- Force mathdir in math if ON (currently works as expected only
8068 -- with 'l')
8069
8070 if inmath and d == 'on' then
8071   d = ('TRT' == tex.mathdir) and 'r' or 'l'
8072 end
8073
8074 if d then
8075   if d == 'al' then
8076     d = 'r'
8077     last = 'al'
8078   elseif d == 'l' or d == 'r' then
8079     last = d

```

```

8080     end
8081     prev_d = d
8082     table.insert(nodes, {item, d, outer_first})
8083 end
8084
8085 outer_first = nil
8086
8087 ::nextnode::
8088
8089 end -- for each node
8090
8091 -- TODO -- repeated here in case EN/ET is the last node. Find a
8092 -- better way of doing things:
8093 if first_et then      -- dir may be nil here !
8094     if has_en then
8095         if last == 'l' then
8096             temp = 'l'    -- W7
8097         else
8098             temp = 'en'  -- W5
8099         end
8100     else
8101         temp = 'on'     -- W6
8102     end
8103     for e = first_et, #nodes do
8104         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8105     end
8106 end
8107
8108 -- dummy node, to close things
8109 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8110
8111 ----- NEUTRAL -----
8112
8113 outer = save_outer
8114 last = outer
8115
8116 local first_on = nil
8117
8118 for q = 1, #nodes do
8119     local item
8120
8121     local outer_first = nodes[q][3]
8122     outer = outer_first or outer
8123     last = outer_first or last
8124
8125     local d = nodes[q][2]
8126     if d == 'an' or d == 'en' then d = 'r' end
8127     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8128
8129     if d == 'on' then
8130         first_on = first_on or q
8131     elseif first_on then
8132         if last == d then
8133             temp = d
8134         else
8135             temp = outer
8136         end
8137         for r = first_on, q - 1 do
8138             nodes[r][2] = temp
8139             item = nodes[r][1]    -- MIRRORING
8140             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8141                 and temp == 'r' and characters[item.char] then
8142                 local font_mode = ''

```

```

8143         if item.font > 0 and font.fonts[item.font].properties then
8144             font_mode = font.fonts[item.font].properties.mode
8145         end
8146         if font_mode ~= 'harf' and font_mode ~= 'plug' then
8147             item.char = characters[item.char].m or item.char
8148         end
8149     end
8150 end
8151     first_on = nil
8152 end
8153
8154     if d == 'r' or d == 'l' then last = d end
8155 end
8156
8157 ----- IMPLICIT, REORDER -----
8158
8159 outer = save_outer
8160 last = outer
8161
8162 local state = {}
8163 state.has_r = false
8164
8165 for q = 1, #nodes do
8166     local item = nodes[q][1]
8167
8168     outer = nodes[q][3] or outer
8169
8170     local d = nodes[q][2]
8171
8172
8173     if d == 'nsm' then d = last end           -- W1
8174     if d == 'en' then d = 'an' end
8175     local isdir = (d == 'r' or d == 'l')
8176
8177     if outer == 'l' and d == 'an' then
8178         state.san = state.san or item
8179         state.ean = item
8180     elseif state.san then
8181         head, state = insert_numeric(head, state)
8182     end
8183
8184     if outer == 'l' then
8185         if d == 'an' or d == 'r' then      -- im -> implicit
8186             if d == 'r' then state.has_r = true end
8187             state.sim = state.sim or item
8188             state.eim = item
8189         elseif d == 'l' and state.sim and state.has_r then
8190             head, state = insert_implicit(head, state, outer)
8191         elseif d == 'l' then
8192             state.sim, state.eim, state.has_r = nil, nil, false
8193         end
8194     else
8195         if d == 'an' or d == 'l' then
8196             if nodes[q][3] then -- nil except after an explicit dir
8197                 state.sim = item -- so we move sim 'inside' the group
8198             else
8199                 state.sim = state.sim or item
8200             end
8201             state.eim = item
8202         elseif d == 'r' and state.sim then
8203             head, state = insert_implicit(head, state, outer)
8204         elseif d == 'r' then
8205             state.sim, state.eim = nil, nil

```

```

8206     end
8207 end
8208
8209 if isdir then
8210     last = d          -- Don't search back - best save now
8211 elseif d == 'on' and state.san then
8212     state.san = state.san or item
8213     state.ean = item
8214 end
8215
8216 end
8217
8218 head = node.prev(head) or head
8219% \end{macrocode}
8220%
8221% Now direction nodes has been distributed with relation to characters
8222% and spaces, we need to take into account \TeX-specific elements in
8223% the node list, to move them at an appropriate place. Firstly, with
8224% hyperlinks. Secondly, we avoid them between penalties and spaces, so
8225% that the latter are still discardable.
8226%
8227% \begin{macrocode}
8228 --- FIXES ---
8229 if has_hyperlink then
8230     local flag, linking = 0, 0
8231     for item in node.traverse(head) do
8232         if item.id == DIR then
8233             if item.dir == '+TRT' or item.dir == '+TLT' then
8234                 flag = flag + 1
8235             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8236                 flag = flag - 1
8237             end
8238             elseif item.id == 8 and item.subtype == 19 then
8239                 linking = flag
8240             elseif item.id == 8 and item.subtype == 20 then
8241                 if linking > 0 then
8242                     if item.prev.id == DIR and
8243                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8244                         d = node.new(DIR)
8245                         d.dir = item.prev.dir
8246                         node.remove(head, item.prev)
8247                         node.insert_after(head, item, d)
8248                     end
8249                 end
8250                 linking = 0
8251             end
8252         end
8253     end
8254
8255     for item in node.traverse_id(10, head) do
8256         local p = item
8257         local flag = false
8258         while p.prev and p.prev.id == 14 do
8259             flag = true
8260             p = p.prev
8261         end
8262         if flag then
8263             node.insert_before(head, p, node.copy(item))
8264             node.remove(head, item)
8265         end
8266     end
8267
8268     return head

```

```

8269 end
8270 function Babel.unset_atdir(head)
8271   local ATDIR = Babel.attr_dir
8272   for item in node.traverse(head) do
8273     node.set_attribute(item, ATDIR, 0x80)
8274   end
8275   return head
8276 end
8277  $\langle$ /basic $\rangle$ 

```

## 11. 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.

## 12. 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.

```

8278  $\langle$ *nil $\rangle$ 
8279 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8280 \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.

```

8281 \ifx\l@nil\undefined
8282   \newlanguage\l@nil
8283   \@namedef{bbl@hyphendata@the\l@nil}{}{}% Remove warning
8284   \let\bbl@elt\relax
8285   \edef\bbl@languages{% Add it to the list of languages
8286     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
8287 \fi

```

This macro is used to store the values of the hyphenation parameters `\leftthyphenmin` and `\rightthyphenmin`.

```

8288 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

**`\captionnil`**

**`\datenil`**

```

8289 \let\captionnil\@empty
8290 \let\datenil\@empty

```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```

8291 \def\bbl@inidata@nil{%
8292   \bbl@elt{identification}{tag.ini}{und}%
8293   \bbl@elt{identification}{load.level}{0}%
8294   \bbl@elt{identification}{charset}{utf8}%
8295   \bbl@elt{identification}{version}{1.0}%

```

```

8296 \bbl@elt{identification}{date}{2022-05-16}%
8297 \bbl@elt{identification}{name.local}{nil}%
8298 \bbl@elt{identification}{name.english}{nil}%
8299 \bbl@elt{identification}{name.babel}{nil}%
8300 \bbl@elt{identification}{tag.bcp47}{und}%
8301 \bbl@elt{identification}{language.tag.bcp47}{und}%
8302 \bbl@elt{identification}{tag.opentype}{dflt}%
8303 \bbl@elt{identification}{script.name}{Latin}%
8304 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8305 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8306 \bbl@elt{identification}{level}{1}%
8307 \bbl@elt{identification}{encodings}{}%
8308 \bbl@elt{identification}{derivate}{no}}
8309 \@namedef{bbl@tbc@nil}{und}
8310 \@namedef{bbl@lbc@nil}{und}
8311 \@namedef{bbl@casing@nil}{und} % TODO
8312 \@namedef{bbl@lotf@nil}{dflt}
8313 \@namedef{bbl@elname@nil}{nil}
8314 \@namedef{bbl@lname@nil}{nil}
8315 \@namedef{bbl@esname@nil}{Latin}
8316 \@namedef{bbl@sname@nil}{Latin}
8317 \@namedef{bbl@sbc@nil}{Latn}
8318 \@namedef{bbl@sotf@nil}{latn}

```

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.

```

8319 \ldf@finish{nil}
8320 </nil>

```

## 13. Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the identification section with `require.calendars`.

Start with function to compute the Julian day. It's based on the little library `calendar.js`, by John Walker, in the public domain.

```

8321 <<{*Compute Julian day}>> ≡
8322 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8323 \def\bbl@cs@gregleap#1{%
8324   (\bbl@fpmo{#1}{4} == 0) &&
8325   (!( (\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0) ) )}
8326 \def\bbl@cs@jd#1#2#3{% year, month, day
8327   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8328     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8329     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8330     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }
8331 <</Compute Julian day>>

```

### 13.1. Islamic

The code for the Civil calendar is based on it, too.

```

8332 <{*ca-islamic}
8333 \ExplSyntaxOn
8334 <@Compute Julian day@>
8335 % == islamic (default)
8336 % Not yet implemented
8337 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8338 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8339   ((#3 + ceil(29.5 * (#2 - 1)) +
8340     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8341     1948439.5) - 1) }

```

```

8342 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8343 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8344 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8345 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8346 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8347 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8348   \edef\bbl@tempa{%
8349     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8350   \edef#5{%
8351     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8352   \edef#6{\fp_eval:n{
8353     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8354   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1 }}

```

The Umm al-Qura calendar, used mainly in Saudi Arabia, is based on moment-hijri, by Abdullah Alsigar (license MIT).

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ~1435/~1460 (Gregorian ~2014/~2038).

```

8355 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8356 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8357 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8358 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8359 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8360 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8361 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8362 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8363 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8364 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8365 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8366 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8367 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8368 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8369 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8370 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8371 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8372 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8373 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8374 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8375 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8376 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8377 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8378 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8379 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8380 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8381 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8382 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8383 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8384 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8385 65401,65431,65460,65490,65520}
8386 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8387 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8388 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8389 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8390   \ifnum#2>2014 \ifnum#2<2038
8391     \bbl@afterfi\expandafter\@gobble
8392   \fi\fi
8393   {\bbl@error{year-out-range}{2014-2038}{}}%
8394   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8395     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8396   \count@\@ne
8397   \bbl@foreach\bbl@cs@umalqura@data{%
8398     \advance\count@\@ne
8399     \ifnum##1>\bbl@tempd\else

```



```

8400     \edef\bbl@tempe{\the\count@}%
8401     \edef\bbl@tempb{##1}%
8402     \fi}%
8403 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8404 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8405 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8406 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8407 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
8408 \ExplSyntaxOff
8409 \bbl@add\bbl@precalendar{%
8410   \bbl@replace\bbl@ld@calendar{-civil}{}}%
8411   \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8412   \bbl@replace\bbl@ld@calendar{+}{}}%
8413   \bbl@replace\bbl@ld@calendar{-}{}}
8414 </ca-islamic>

```

## 13.2. Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptations by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in `hebcald.sty`

```

8415 < *ca-hebrew >
8416 \newcount\bbl@cntcommon
8417 \def\bbl@remainder#1#2#3{%
8418   #3=#1\relax
8419   \divide #3 by #2\relax
8420   \multiply #3 by -#2\relax
8421   \advance #3 by #1\relax}%
8422 \newif\ifbbl@divisible
8423 \def\bbl@checkifdivisible#1#2{%
8424   {\countdef\tmp=0
8425     \bbl@remainder{#1}{#2}{\tmp}%
8426     \ifnum \tmp=0
8427       \global\bbl@divisibletrue
8428     \else
8429       \global\bbl@divisiblefalse
8430     \fi}}
8431 \newif\ifbbl@gregleap
8432 \def\bbl@ifgregleap#1{%
8433   \bbl@checkifdivisible{#1}{4}%
8434   \ifbbl@divisible
8435     \bbl@checkifdivisible{#1}{100}%
8436     \ifbbl@divisible
8437       \bbl@checkifdivisible{#1}{400}%
8438       \ifbbl@divisible
8439         \bbl@gregleaptrue
8440       \else
8441         \bbl@gregleapfalse
8442       \fi
8443     \else
8444       \bbl@gregleaptrue
8445     \fi
8446   \else
8447     \bbl@gregleapfalse
8448   \fi
8449   \ifbbl@gregleap}
8450 \def\bbl@gregdayspriormonths#1#2#3{%
8451   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8452     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8453   \bbl@ifgregleap{#2}%
8454   \ifnum #1 > 2
8455     \advance #3 by 1
8456   \fi

```

```

8457     \fi
8458     \global\bbl@cntcommon=#3}%
8459     #3=\bbl@cntcommon}
8460 \def\bbl@gregdaysprioryears#1#2{%
8461   {\countdef\tmpc=4
8462    \countdef\tmpb=2
8463    \tmpb=#1\relax
8464    \advance \tmpb by -1
8465    \tmpc=\tmpb
8466    \multiply \tmpc by 365
8467    #2=\tmpc
8468    \tmpc=\tmpb
8469    \divide \tmpc by 4
8470    \advance #2 by \tmpc
8471    \tmpc=\tmpb
8472    \divide \tmpc by 100
8473    \advance #2 by -\tmpc
8474    \tmpc=\tmpb
8475    \divide \tmpc by 400
8476    \advance #2 by \tmpc
8477    \global\bbl@cntcommon=#2\relax}%
8478     #2=\bbl@cntcommon}
8479 \def\bbl@absfromgreg#1#2#3#4{%
8480   {\countdef\tmpd=0
8481    #4=#1\relax
8482    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8483    \advance #4 by \tmpd
8484    \bbl@gregdaysprioryears{#3}{\tmpd}%
8485    \advance #4 by \tmpd
8486    \global\bbl@cntcommon=#4\relax}%
8487     #4=\bbl@cntcommon}
8488 \newif\ifbbl@hebrleap
8489 \def\bbl@checkleaphebryear#1{%
8490   {\countdef\tmpa=0
8491    \countdef\tmpb=1
8492    \tmpa=#1\relax
8493    \multiply \tmpa by 7
8494    \advance \tmpa by 1
8495    \bbl@remainder{\tmpa}{19}{\tmpb}%
8496    \ifnum \tmpb < 7
8497      \global\bbl@hebrleaptrue
8498    \else
8499      \global\bbl@hebrleapfalse
8500    \fi}}
8501 \def\bbl@hebrleapsedmonths#1#2{%
8502   {\countdef\tmpa=0
8503    \countdef\tmpb=1
8504    \countdef\tmpc=2
8505    \tmpa=#1\relax
8506    \advance \tmpa by -1
8507    #2=\tmpa
8508    \divide #2 by 19
8509    \multiply #2 by 235
8510    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8511    \tmpc=\tmpb
8512    \multiply \tmpb by 12
8513    \advance #2 by \tmpb
8514    \multiply \tmpc by 7
8515    \advance \tmpc by 1
8516    \divide \tmpc by 19
8517    \advance #2 by \tmpc
8518    \global\bbl@cntcommon=#2}%
8519     #2=\bbl@cntcommon}

```

```

8520 \def\bbl@hebreleaseddays#1#2{%
8521   {\countdef\tmpa=0
8522    \countdef\tmpb=1
8523    \countdef\tmpc=2
8524    \bbl@hebreleasedmonths{#1}{#2}%
8525    \tmpa=#2\relax
8526    \multiply \tmpa by 13753
8527    \advance \tmpa by 5604
8528    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8529    \divide \tmpa by 25920
8530    \multiply #2 by 29
8531    \advance #2 by 1
8532    \advance #2 by \tmpa
8533    \bbl@remainder{#2}{7}{\tmpa}%
8534    \ifnum \tmpc < 19440
8535      \ifnum \tmpc < 9924
8536        \else
8537          \ifnum \tmpa=2
8538            \bbl@checkleaphebrewyear{#1}% of a common year
8539            \ifbbl@hebrleap
8540              \else
8541                \advance #2 by 1
8542              \fi
8543            \fi
8544          \fi
8545          \ifnum \tmpc < 16789
8546            \else
8547              \ifnum \tmpa=1
8548                \advance #1 by -1
8549                \bbl@checkleaphebrewyear{#1}% at the end of leap year
8550                \ifbbl@hebrleap
8551                  \advance #2 by 1
8552                \fi
8553              \fi
8554            \fi
8555          \else
8556            \advance #2 by 1
8557          \fi
8558          \bbl@remainder{#2}{7}{\tmpa}%
8559          \ifnum \tmpa=0
8560            \advance #2 by 1
8561          \else
8562            \ifnum \tmpa=3
8563              \advance #2 by 1
8564            \else
8565              \ifnum \tmpa=5
8566                \advance #2 by 1
8567              \fi
8568            \fi
8569          \fi
8570          \global\bbl@cntcommon=#2\relax}%
8571   #2=\bbl@cntcommon}
8572 \def\bbl@daysinhebrewyear#1#2{%
8573   {\countdef\tmpe=12
8574    \bbl@hebreleaseddays{#1}{\tmpe}%
8575    \advance #1 by 1
8576    \bbl@hebreleaseddays{#1}{#2}%
8577    \advance #2 by -\tmpe
8578    \global\bbl@cntcommon=#2}%
8579   #2=\bbl@cntcommon}
8580 \def\bbl@hebrdayspriormonths#1#2#3{%
8581   {\countdef\tmpf= 14
8582    #3=\ifcase #1

```

```

8583         0 \or
8584         0 \or
8585         30 \or
8586         59 \or
8587         89 \or
8588         118 \or
8589         148 \or
8590         148 \or
8591         177 \or
8592         207 \or
8593         236 \or
8594         266 \or
8595         295 \or
8596         325 \or
8597         400
8598     \fi
8599     \bbl@checkleaphebrewyear{#2}%
8600     \ifbbl@hebrleap
8601         \ifnum #1 > 6
8602             \advance #3 by 30
8603         \fi
8604     \fi
8605     \bbl@daysinhebrewyear{#2}{\tmpf}%
8606     \ifnum #1 > 3
8607         \ifnum \tmpf=353
8608             \advance #3 by -1
8609         \fi
8610         \ifnum \tmpf=383
8611             \advance #3 by -1
8612         \fi
8613     \fi
8614     \ifnum #1 > 2
8615         \ifnum \tmpf=355
8616             \advance #3 by 1
8617         \fi
8618         \ifnum \tmpf=385
8619             \advance #3 by 1
8620         \fi
8621     \fi
8622     \global\bbl@cntcommon=#3\relax}%
8623     #3=\bbl@cntcommon}
8624 \def\bbl@absfromhebr#1#2#3#4{%
8625     {#4=#1\relax
8626     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8627     \advance #4 by #1\relax
8628     \bbl@hebrrelapseddays{#3}{#1}%
8629     \advance #4 by #1\relax
8630     \advance #4 by -1373429
8631     \global\bbl@cntcommon=#4\relax}%
8632     #4=\bbl@cntcommon}
8633 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8634     {\countdef\tmpx= 17
8635     \countdef\tmpy= 18
8636     \countdef\tmpz= 19
8637     #6=#3\relax
8638     \global\advance #6 by 3761
8639     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8640     \tmpz=1 \tmpy=1
8641     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8642     \ifnum \tmpx > #4\relax
8643         \global\advance #6 by -1
8644         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8645     \fi

```

```

8646 \advance #4 by -\tmpx
8647 \advance #4 by 1
8648 #5=#4\relax
8649 \divide #5 by 30
8650 \loop
8651 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8652 \ifnum \tmpx < #4\relax
8653 \advance #5 by 1
8654 \tmpy=\tmpx
8655 \repeat
8656 \global\advance #5 by -1
8657 \global\advance #4 by -\tmpy}}
8658 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8659 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8660 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8661 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8662 \bbl@hebrfromgreg
8663 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8664 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8665 \edef#4{\the\bbl@hebryear}%
8666 \edef#5{\the\bbl@hebrmonth}%
8667 \edef#6{\the\bbl@hebrday}}
8668 </ca-hebrew>

```

### 13.3. Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use with LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```

8669 <*ca-persian>
8670 \ExplSyntaxOn
8671 <@Compute Julian day@>
8672 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8673 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8674 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8675 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8676 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8677 \bbl@afterfi\expandafter@gobble
8678 \fi\fi
8679 {\bbl@error{year-out-range}{2013-2050}{}}}%
8680 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8681 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8682 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8683 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8684 \ifnum\bbl@tempc<\bbl@tempb
8685 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8686 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8687 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8688 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8689 \fi
8690 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8691 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8692 \edef#5{\fp_eval:n{% set Jalali month
8693 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8694 \edef#6{\fp_eval:n{% set Jalali day
8695 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
8696 \ExplSyntaxOff
8697 </ca-persian>

```

## 13.4. Coptic and Ethiopic

Adapted from `jquery.calendars.package-1.1.4`, written by Keith Wood, 2010. Dual license: GPL and MIT. The only difference is the epoch.

```
8698 (*ca-coptic)
8699 \ExplSyntaxOn
8700 <@Compute Julian day@>
8701 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8702 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8703 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8704 \edef#4{\fp_eval:n{%
8705 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8706 \edef\bbl@tempc{\fp_eval:n{%
8707 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8708 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8709 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8710 \ExplSyntaxOff
8711 </ca-coptic>
8712 (*ca-ethiopic)
8713 \ExplSyntaxOn
8714 <@Compute Julian day@>
8715 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8716 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8717 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8718 \edef#4{\fp_eval:n{%
8719 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8720 \edef\bbl@tempc{\fp_eval:n{%
8721 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8722 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8723 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8724 \ExplSyntaxOff
8725 </ca-ethiopic>
```

## 13.5. Buddhist

That's very simple.

```
8726 (*ca-buddhist)
8727 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8728 \edef#4{\number\numexpr#1+543\relax}%
8729 \edef#5{#2}%
8730 \edef#6{#3}}
8731 </ca-buddhist>
8732 %
8733 % \subsection{Chinese}
8734 %
8735 % Brute force, with the Julian day of first day of each month. The
8736 % table has been computed with the help of \textsf{python-lunardate} by
8737 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8738 % is 2015-2044.
8739 %
8740 % \begin{macrocode}
8741 (*ca-chinese)
8742 \ExplSyntaxOn
8743 <@Compute Julian day@>
8744 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
8745 \edef\bbl@tempd{\fp_eval:n{%
8746 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8747 \count@z@
8748 \@tempcnta=2015
8749 \bbl@foreach\bbl@cs@chinese@data{%
8750 \ifnum##1>\bbl@tempd\else
8751 \advance\count@\@ne
8752 \ifnum\count@>12
```

```

8753     \count@\@ne
8754     \advance\@tempcnta\@ne\fi
8755     \bbl@xin@{,##1,},{,\bbl@cs@chinese@leap,}%
8756     \ifin@
8757     \advance\count@\m@ne
8758     \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8759     \else
8760     \edef\bbl@tempe{\the\count@}%
8761     \fi
8762     \edef\bbl@tempb{##1}%
8763     \fi}%
8764 \edef#4{\the\@tempcnta}%
8765 \edef#5{\bbl@tempe}%
8766 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8767 \def\bbl@cs@chinese@leap{%
8768   885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8769 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8770   354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8771   768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8772   1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8773   1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8774   1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8775   2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8776   2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8777   2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8778   3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8779   3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8780   3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8781   4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8782   4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8783   5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8784   5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8785   5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8786   6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8787   6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8788   6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8789   7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8790   7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8791   7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8792   8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8793   8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8794   8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8795   9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8796   9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8797   10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8798   10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8799   10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8800   10896,10926,10956,10986,11015,11045,11074,11103}
8801 \ExplSyntaxOff
8802 </ca-chinese>

```

## 14. Support for Plain T<sub>E</sub>X (plain.def)

### 14.1. Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T<sub>E</sub>X-format. When asked he responded:

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

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTeX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing `iniTeX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```
8803 <(*bplain | blplain)
8804 \catcode`\{=1 % left brace is begin-group character
8805 \catcode`\}=2 % right brace is end-group character
8806 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
8807 \openin 0 hyphen.cfg
8808 \ifeof0
8809 \else
8810 \let\a\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead. Once that's done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
8811 \def\input #1 {%
8812 \let\input\a
8813 \a hyphen.cfg
8814 \let\a\undefined
8815 }
8816 \fi
8817 </bplain | blplain>
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
8818 <bplain>\a plain.tex
8819 <blplain>\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
8820 <bplain>\def\fmtname{babel-plain}
8821 <blplain>\def\fmtname{babel-lplain}
```

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

## 14.2. Emulating some $\LaTeX$ features

The file `babel.def` expects some definitions made in the  $\LaTeX 2_{\epsilon}$  style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading `babel`. `\BabelModifiers` can be set too (but not sure it works).

```
8822 <<(*Emulate LaTeX)>> ≡
8823 \def\@empty{}
8824 \def\loadlocalcfg#1{%
8825 \openin0#1.cfg
8826 \ifeof0
8827 \closein0
8828 \else
8829 \closein0
8830 {\immediate\writel6{*****}%
8831 \immediate\writel6{* Local config file #1.cfg used}%
8832 \immediate\writel6{*}%
8833 }
8834 \input #1.cfg\relax
8835 \fi
8836 \@endofldf}
```



## 14.3. General tools

A number of  $\TeX$  macro's that are needed later on.

```
8837 \long\def\@firstofone#1{#1}
8838 \long\def\@firstoftwo#1#2{#1}
8839 \long\def\@secondoftwo#1#2{#2}
8840 \def\@nnil{\@nil}
8841 \def\@gobbletwo#1#2{}
8842 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8843 \def\@star@or@long#1{%
8844   \@ifstar
8845   {\let\@ngrel@x\relax#1}%
8846   {\let\@ngrel@x\long#1}}
8847 \let\@ngrel@x\relax
8848 \def\@car#1#2\@nil{#1}
8849 \def\@cdr#1#2\@nil{#2}
8850 \let\@typeset@protect\relax
8851 \let\protected@edef\edef
8852 \long\def\@gobble#1{}
8853 \edef\@backslashchar{\expandafter\@gobble\string\}
8854 \def\strip@prefix#1>{}
8855 \def\g@addto@macro#1#2{{%
8856   \toks@\expandafter{#1#2}%
8857   \xdef#1{\the\toks@}}}
8858 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8859 \def\@nameuse#1{\csname #1\endcsname}
8860 \def\@ifundefined#1{%
8861   \expandafter\ifx\csname#1\endcsname\relax
8862     \expandafter\@firstoftwo
8863     \else
8864     \expandafter\@secondoftwo
8865     \fi}
8866 \def\@expandtwoargs#1#2#3{%
8867   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8868 \def\zap@space#1 #2{%
8869   #1%
8870   \ifx#2\@empty\else\expandafter\zap@space\fi
8871   #2}
8872 \let\bbl@trace\@gobble
8873 \def\bbl@error#1{% Implicit #2#3#4
8874   \begingroup
8875     \catcode`\=0 \catcode`\==12 \catcode`\`=12
8876     \catcode`\^M=5 \catcode`\%=14
8877     \input errbabel.def
8878   \endgroup
8879   \bbl@error{#1}}
8880 \def\bbl@warning#1{%
8881   \begingroup
8882     \newlinechar=`^^J
8883     \def\{\^J(babel) }%
8884     \message{\#1}%
8885   \endgroup}
8886 \let\bbl@infowarn\bbl@warning
8887 \def\bbl@info#1{%
8888   \begingroup
8889     \newlinechar=`^^J
8890     \def\{\^J}%
8891     \wlog{#1}%
8892   \endgroup}
8893 \ifx\@preamblecmds\undefined
```

$\TeX 2\epsilon$  has the command `\onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

8894 \def\@preamblecmds{}
8895 \fi
8896 \def\@onlypreamble#1{%
8897 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8898 \@preamblecmds\do#1}}
8899 \@onlypreamble\@onlypreamble

```

Mimic L<sup>A</sup>T<sub>E</sub>X's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```

8900 \def\begindocument{%
8901 \@begindocumenthook
8902 \global\let\@begindocumenthook\@undefined
8903 \def\do##1{\global\let##1\@undefined}%
8904 \@preamblecmds
8905 \global\let\do\noexpand}

8906 \ifx\@begindocumenthook\@undefined
8907 \def\@begindocumenthook{}
8908 \fi
8909 \@onlypreamble\@begindocumenthook
8910 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimic L<sup>A</sup>T<sub>E</sub>X's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```

8911 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8912 \@onlypreamble\AtEndOfPackage
8913 \def\@endofldf{}
8914 \@onlypreamble\@endofldf
8915 \let\bbl@afterlang\@empty
8916 \chardef\bbl@opt@hyphenmap\z@

```

L<sup>A</sup>T<sub>E</sub>X 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.

```

8917 \catcode`\&=\z@
8918 \ifx&\if@filesw\@undefined
8919 \expandafter\let\csname if@filesw\expandafter\endcsname
8920 \csname iffalse\endcsname
8921 \fi
8922 \catcode`\&=4

```

Mimic L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

8923 \def\newcommand{\@star@or@long\new@command}
8924 \def\new@command#1{%
8925 \@testopt{\@newcommand#1}0}
8926 \def\@newcommand#1[#2]{%
8927 \@ifnextchar [ {\@xargdef#1[#2]}%
8928 \{\@argdef#1[#2]}}
8929 \long\def\@argdef#1[#2]#3{%
8930 \@yargdef#1\@ne{#2}{#3}}
8931 \long\def\@xargdef#1[#2][#3]#4{%
8932 \expandafter\def\expandafter#1\expandafter{%
8933 \expandafter\@protected@testopt\expandafter #1%
8934 \csname\string#1\expandafter\endcsname{#3}}%
8935 \expandafter\@yargdef \csname\string#1\endcsname
8936 \tw@{#2}{#4}}
8937 \long\def\@yargdef#1#2#3{%
8938 \@tempcnta#3\relax
8939 \advance \@tempcnta \@ne
8940 \let\@hash@\relax
8941 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8942 \@tempcntb #2%
8943 \@whilenum\@tempcntb <\@tempcnta
8944 \do{%
8945 \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%

```

```

8946 \advance\@tempcntb \@ne}%
8947 \let\@hash###%
8948 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8949 \def\providecommand{\@star@or@long\provide@command}
8950 \def\provide@command#1{%
8951 \begingroup
8952 \escapechar\m@ne\xdef\@gtempa{\string#1}%
8953 \endgroup
8954 \expandafter\ifundefined\@gtempa
8955 {\def\reserved@a{\new@command#1}}%
8956 {\let\reserved@a\relax
8957 \def\reserved@a{\new@command\reserved@a}}%
8958 \reserved@a}%

8959 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8960 \def\declare@robustcommand#1{%
8961 \edef\reserved@a{\string#1}%
8962 \def\reserved@b{#1}%
8963 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8964 \edef#1{%
8965 \ifx\reserved@a\reserved@b
8966 \noexpand\x@protect
8967 \noexpand#1%
8968 \fi
8969 \noexpand\protect
8970 \expandafter\noexpand\csname
8971 \expandafter\@gobble\string#1 \endcsname
8972 }%
8973 \expandafter\new@command\csname
8974 \expandafter\@gobble\string#1 \endcsname
8975 }
8976 \def\x@protect#1{%
8977 \ifx\protect\@typeset@protect\else
8978 \x@protect#1%
8979 \fi
8980 }
8981 \catcode`\&=\z@ % Trick to hide conditionals
8982 \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`.

```

8983 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8984 \catcode`\&=4
8985 \ifx\in@\@undefined
8986 \def\in@#1#2{%
8987 \def\in@##1#1##2##3\in@@{%
8988 \ifx\in@##2\in@false\else\in@true\fi}%
8989 \in@@#2#1\in@\in@@}
8990 \else
8991 \let\bbl@tempa\@empty
8992 \fi
8993 \bbl@tempa

```

$\TeX$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\TeX$  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).

```

8994 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\TeX$  macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\TeX$  but we need the macro to be defined as a no-op.

```

8995 \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  $\TeX 2\epsilon$  versions; just enough to make things work in plain  $\TeX$  environments.

```
8996 \ifx\@tempcnta\@undefined
8997   \csname newcount\endcsname\@tempcnta\relax
8998 \fi
8999 \ifx\@tempcntb\@undefined
9000   \csname newcount\endcsname\@tempcntb\relax
9001 \fi
```

To prevent wasting two counters in  $\TeX$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
9002 \ifx\bye\@undefined
9003   \advance\count10 by -2\relax
9004 \fi
9005 \ifx\@ifnextchar\@undefined
9006   \def\@ifnextchar#1#2#3{%
9007     \let\reserved@d=#1%
9008     \def\reserved@a{#2}\def\reserved@b{#3}%
9009     \futurelet\@let@token\@ifnch}
9010   \def\@ifnch{%
9011     \ifx\@let@token\@sptoken
9012       \let\reserved@c\@xifnch
9013     \else
9014       \ifx\@let@token\reserved@d
9015         \let\reserved@c\reserved@a
9016       \else
9017         \let\reserved@c\reserved@b
9018       \fi
9019     \fi
9020     \reserved@c}
9021   \def\{\let\@sptoken= } \: % this makes \@sptoken a space token
9022   \def\{\@xifnch} \expandafter\def\{\futurelet\@let@token\@ifnch}
9023 \fi
9024 \def\@testopt#1#2{%
9025   \@ifnextchar[#{1}{#1[#2]}}
9026 \def\@protected@testopt#1{%
9027   \ifx\protect\@typeset@protect
9028     \expandafter\@testopt
9029   \else
9030     \@x@protect#1%
9031   \fi}
9032 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9033   #2\relax}\fi}
9034 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9035   \else\expandafter\@gobble\fi{#1}}
```

## 14.4. Encoding related macros

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

```
9036 \def\DeclareTextCommand{%
9037   \@dec@text@cmd\providecommand
9038 }
9039 \def\ProvideTextCommand{%
9040   \@dec@text@cmd\providecommand
9041 }
9042 \def\DeclareTextSymbol#1#2#3{%
9043   \@dec@text@cmd\chardef#1{#2}#3\relax
9044 }
9045 \def\@dec@text@cmd#1#2#3{%
9046   \expandafter\def\expandafter#2%
9047     \expandafter{%
```

```

9048     \csname#3-cmd\expandafter\endcsname
9049     \expandafter#2%
9050     \csname#3\string#2\endcsname
9051     }%
9052 % \let@ifdefinable\@rc@ifdefinable
9053 \expandafter#1\csname#3\string#2\endcsname
9054 }
9055 \def\@current@cmd#1{%
9056 \ifx\protect\@typeset@protect\else
9057 \noexpand#1\expandafter\@gobble
9058 \fi
9059 }
9060 \def\@changed@cmd#1#2{%
9061 \ifx\protect\@typeset@protect
9062 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9063 \expandafter\ifx\csname ?\string#1\endcsname\relax
9064 \expandafter\def\csname ?\string#1\endcsname{%
9065 \@changed@x@err{#1}%
9066 }%
9067 \fi
9068 \global\expandafter\let
9069 \csname\cf@encoding \string#1\expandafter\endcsname
9070 \csname ?\string#1\endcsname
9071 \fi
9072 \csname\cf@encoding\string#1%
9073 \expandafter\endcsname
9074 \else
9075 \noexpand#1%
9076 \fi
9077 }
9078 \def\@changed@x@err#1{%
9079 \errhelp{Your command will be ignored, type <return> to proceed}%
9080 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9081 \def\DeclareTextCommandDefault#1{%
9082 \DeclareTextCommand#1?%
9083 }
9084 \def\ProvideTextCommandDefault#1{%
9085 \ProvideTextCommand#1?%
9086 }
9087 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9088 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9089 \def\DeclareTextAccent#1#2#3{%
9090 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
9091 }
9092 \def\DeclareTextCompositeCommand#1#2#3#4{%
9093 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9094 \edef\reserved@b{\string##1}%
9095 \edef\reserved@c{%
9096 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9097 \ifx\reserved@b\reserved@c
9098 \expandafter\expandafter\expandafter\ifx
9099 \expandafter\@car\reserved@a\relax\relax\@nil
9100 \@text@composite
9101 \else
9102 \edef\reserved@b##1{%
9103 \def\expandafter\noexpand
9104 \csname#2\string#1\endcsname###1{%
9105 \noexpand\@text@composite
9106 \expandafter\noexpand\csname#2\string#1\endcsname
9107 ###1\noexpand\@empty\noexpand\@text@composite
9108 {##1}%
9109 }%
9110 }%

```

```

9111     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9112     \fi
9113     \expandafter\def\csname\expandafter\string\csname
9114       #2\endcsname\string#1-\string#3\endcsname{#4}
9115   \else
9116     \errhelp{Your command will be ignored, type <return> to proceed}%
9117     \errmessage{\string\DeclareTextCompositeCommand\space used on
9118       inappropriate command \protect#1}
9119   \fi
9120 }
9121 \def\@text@composite#1#2#3\@text@composite{%
9122   \expandafter\@text@composite@x
9123     \csname\string#1-\string#2\endcsname
9124 }
9125 \def\@text@composite@x#1#2{%
9126   \ifx#1\relax
9127     #2%
9128   \else
9129     #1%
9130   \fi
9131 }
9132 %
9133 \def\@strip@args#1:#2-#3\@strip@args{#2}
9134 \def\DeclareTextComposite#1#2#3#4{%
9135   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9136   \bgroup
9137     \lccode`\@=#4%
9138     \lowercase{%
9139   \egroup
9140     \reserved@a @%
9141   }%
9142 }
9143 %
9144 \def\UseTextSymbol#1#2{#2}
9145 \def\UseTextAccent#1#2#3{}
9146 \def\@use@text@encoding#1{}
9147 \def\DeclareTextSymbolDefault#1#2{%
9148   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9149 }
9150 \def\DeclareTextAccentDefault#1#2{%
9151   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9152 }
9153 \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.

```

9154 \DeclareTextAccent{"}{OT1}{127}
9155 \DeclareTextAccent{'}{OT1}{19}
9156 \DeclareTextAccent{^}{OT1}{94}
9157 \DeclareTextAccent{\`}{OT1}{18}
9158 \DeclareTextAccent{\~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for PLAIN  $\text{\TeX}$ .

```

9159 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9160 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`}
9161 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`'}
9162 \DeclareTextSymbol{\textquoteright}{OT1}{`\`'}
9163 \DeclareTextSymbol{\i}{OT1}{16}
9164 \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`.

```

9165 \ifx\scriptsize\undefined
9166   \let\scriptsize\sevenrm

```

```

9167 \fi
    And a few more “dummy” definitions.
9168 \def\language{english}%
9169 \let\bblopt@shorthands\@nnil
9170 \def\bblopt@ifshorthand#1#2#3{#2}%
9171 \let\bblopt@language@opts\@empty
9172 \let\bblopt@ensureinfo\@gobble
9173 \let\bblopt@provide@locale\relax
9174 \ifx\babeloptionstrings\undefined
9175   \let\bblopt@strings\@nnil
9176 \else
9177   \let\bblopt@strings\babeloptionstrings
9178 \fi
9179 \def\BabelStringsDefault{generic}
9180 \def\bblopt@tempa{normal}
9181 \ifx\babeloptionmath\bblopt@tempa
9182   \def\bblopt@mathnormal{\noexpand\textormath}
9183 \fi
9184 \def\AfterBabelLanguage#1#2{}
9185 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9186 \let\bblopt@afterlang\relax
9187 \def\bblopt@opt@safe{BR}
9188 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
9189 \ifx\bblopt@trace\undefined\def\bblopt@trace#1{}\fi
9190 \expandafter\newif\csname ifbblopt@single\endcsname
9191 \chardef\bblopt@bidimode\z@
9192 <</Emulate LaTeX>>
    A proxy file:
9193 <*\plain>
9194 \input babel.def
9195 </\plain>

```

## 15. Acknowledgements

In the initial stages of the development of babel, Bernd Raichle provided many helpful suggestions and Michel Goossens supplied contributions for many languages. Ideas from Nico Poppelier, Piet van Oostrum and many others have been used. Paul Wackers and Werenfried Spit helped find and repair bugs.

More recently, there are significant contributions by Salim Bou, Ulrike Fischer, Loren Davis and Udi Fogiel.

Barbara Beeton has helped in improving the manual.

There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\TeX$  styles*, *TUGboat* 10 (1989) #3, pp. 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,  *$\TeX$ , 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] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018

- [10] Hubert Partl, *German T<sub>E</sub>X*, *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International L<sup>A</sup>T<sub>E</sub>X is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L<sup>A</sup>T<sub>E</sub>X*, Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).