

# Babel

Code

Version 24.7.58672  
2024/08/08

Javier Bezos  
Current maintainer

Johannes L. Braams  
Original author

Localization and  
internationalization

Unicode

TeX

pdfTeX

LuaTeX

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	Multiple languages . . . . .	7
3.2	The Package File ( <code>\LaTeX</code> , <code>babel.sty</code> ) . . . . .	8
3.3	<code>base</code> . . . . .	9
3.4	<code>key=value</code> options and other general option . . . . .	10
3.5	Conditional loading of shorthands . . . . .	11
3.6	Interlude for Plain . . . . .	13
<b>4</b>	<b>Multiple languages</b>	<b>13</b>
4.1	Selecting the language . . . . .	15
4.2	Errors . . . . .	23
4.3	Hooks . . . . .	25
4.4	Setting up language files . . . . .	27
4.5	Shorthands . . . . .	29
4.6	Language attributes . . . . .	38
4.7	Support for saving macro definitions . . . . .	39
4.8	Short tags . . . . .	41
4.9	Hyphens . . . . .	41
4.10	Multiencoding strings . . . . .	43
4.11	Macros common to a number of languages . . . . .	48
4.12	Making glyphs available . . . . .	48
4.12.1	Quotation marks . . . . .	48
4.12.2	Letters . . . . .	50
4.12.3	Shorthands for quotation marks . . . . .	50
4.12.4	Umlauts and tremas . . . . .	51
4.13	Layout . . . . .	52
4.14	Load engine specific macros . . . . .	53
4.15	Creating and modifying languages . . . . .	53
<b>5</b>	<b>Adjusting the Babel behavior</b>	<b>76</b>
5.1	Cross referencing macros . . . . .	79
5.2	Marks . . . . .	81
5.3	Preventing clashes with other packages . . . . .	82
5.3.1	<code>ifthen</code> . . . . .	82
5.3.2	<code>varioref</code> . . . . .	83
5.3.3	<code>hhline</code> . . . . .	83
5.4	Encoding and fonts . . . . .	84
5.5	Basic bidi support . . . . .	85
5.6	Local Language Configuration . . . . .	89
5.7	Language options . . . . .	89
<b>6</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>92</b>
<b>7</b>	<b>Loading hyphenation patterns</b>	<b>96</b>
<b>8</b>	<b>Font handling with <code>fontspec</code></b>	<b>100</b>
<b>9</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>103</b>
9.1	XeTeX . . . . .	103

<b>10</b>	<b>Support for interchar</b>	<b>105</b>
10.1	Layout	107
10.2	8-bit TeX	109
10.3	LuaTeX	110
10.4	Southeast Asian scripts	116
10.5	CJK line breaking	117
10.6	Arabic justification	119
10.7	Common stuff	124
10.8	Automatic fonts and ids switching	124
10.9	Bidi	130
10.10	Layout	132
10.11	Lua: transforms	140
10.12	Lua: Auto bidi with <code>basic</code> and <code>basic-r</code>	149
<b>11</b>	<b>Data for CJK</b>	<b>160</b>
<b>12</b>	<b>The ‘nil’ language</b>	<b>160</b>
<b>13</b>	<b>Calendars</b>	<b>161</b>
13.1	Islamic	162
13.2	Hebrew	163
13.3	Persian	167
13.4	Coptic and Ethiopic	168
13.5	Buddhist	168
<b>14</b>	<b>Support for Plain TeX (<code>plain.def</code>)</b>	<b>170</b>
14.1	Not renaming <code>hyphen.tex</code>	170
14.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features	171
14.3	General tools	171
14.4	Encoding related macros	175
<b>15</b>	<b>Acknowledgements</b>	<b>177</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

*Code documentation is still under revision.*

The babel package after unpacking consists of the following files:

**babel.sty** is the  $\text{\LaTeX}$  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 (eg, 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 250 languages. They are distributed as a separate zip file, not packed as dtx. Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, 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=24.7.58672>>
2 <<date=2024/08/08>>
```

**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  $\text{\LaTeX}$  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@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
16 \def\bbl@cs#1{\csname bbl@#1\endcsname}
17 \def\bbl@c#1{\csname bbl@#1\language\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` 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`, `\<.>` 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}}

```

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

<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.

memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

```

56 \beginingroup
57   \gdef\bbl@ifunset#1{%
58     \expandafter\ifx\csname#1\endcsname\relax
59       \expandafter\@firstoftwo
60     \else
61       \expandafter\@secondoftwo
62     \fi}
63 \bbl@ifunset{ifcsname}%
64 {}%
65 {\gdef\bbl@ifunset#1{%
66   \ifcsname#1\endcsname
67     \expandafter\ifx\csname#1\endcsname\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, ie, 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{\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` (ie, 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}}

```

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

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}
103 \def\bbl@replace@aux##1#2##2#2{%

```

```

104 \ifx\bbl@nil##2%
105 \toks@{\expandafter{\the\toks@##1}%
106 \else
107 \toks@{\expandafter{\the\toks@##1#3}%
108 \bbl@afterfi
109 \bbl@replace@aux##2#2%
110 \fi}%
111 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112 \edef#1{\the\toks@}}

```

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

```

113 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
114 \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
115 \def\bbl@tempa{#1}%
116 \def\bbl@tempb{#2}%
117 \def\bbl@tempe{#3}}
118 \def\bbl@sreplace#1#2#3{%
119 \begingroup
120 \expandafter\bbl@parsedef\meaning#1\relax
121 \def\bbl@tempc{#2}%
122 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
123 \def\bbl@tempd{#3}%
124 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
125 \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
126 \ifin@
127 \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
128 \def\bbl@tempc{% Expanded an executed below as 'uplevel'
129 \\makeatletter % "internal" macros with @ are assumed
130 \\scantokens{%
131 \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132 \catcode64=\the\catcode64\relax}% Restore @
133 \else
134 \let\bbl@tempc\empty % Not \relax
135 \fi
136 \bbl@exp{% For the 'uplevel' assignments
137 \endgroup
138 \bbl@tempc}} % empty or expand to set #1 with changes
139 \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 pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

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

```

```

156 \else
157 \tw@
158 \fi
159 \else
160 \@ne
161 \fi

```

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

```

162 \def\bbl@bsphack{%
163 \ifhmode
164 \hskip\z@skip
165 \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166 \else
167 \let\bbl@esphack\@empty
168 \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`.

```

169 \def\bbl@cased{%
170 \ifx\oe\OE
171 \expandafter\in@\expandafter
172 {\expandafter\OE\expandafter}\expandafter{\oe}%
173 \ifin@
174 \bbl@afterelse\expandafter\MakeUppercase
175 \else
176 \bbl@afterfi\expandafter\MakeLowercase
177 \fi
178 \else
179 \expandafter\@firstofone
180 \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`).

```

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

```

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

```

192 <<(*Make sure ProvidesFile is defined)>> ≡
193 \ifx\ProvidesFile\@undefined
194 \def\ProvidesFile#1[#2 #3 #4]{%
195 \wlog{File: #1 #4 #3 <#2>}%
196 \let\ProvidesFile\@undefined}
197 \fi
198 <</Make sure ProvidesFile is defined>>

```

### 3.1 Multiple languages

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

```

199 <<(*Define core switching macros)>> ≡

```



```

200 \ifx\language\@undefined
201   \csname newcount\endcsname\language
202 \fi
203 <</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.

```

204 <<*Define core switching macros>> ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 <</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 The Package File ( $\LaTeX$ , `babel.sty`)

```

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

```

Start with some “private” debugging tool, and then define macros for errors.

```

211 \@ifpackagewith{babel}{debug}
212   {\providecommand\bbbl@trace[1]{\message{^^J[ #1 ]}}%
213    \let\bbbl@debug\@firstofone
214    \ifx\directlua\@undefined\else
215      \directlua{ Babel = Babel or {}
216        Babel.debug = true }%
217      \input{babel-debug.tex}%
218    \fi}
219 {\providecommand\bbbl@trace[1]{}%
220  \let\bbbl@debug@gobble
221  \ifx\directlua\@undefined\else
222    \directlua{ Babel = Babel or {}
223      Babel.debug = false }%
224  \fi}
225 \def\bbbl@error#1{% Implicit #2#3#4
226   \begingroup
227     \catcode`\=0 \catcode`\==12 \catcode`\`=12
228     \input errbabel.def
229   \endgroup
230   \bbbl@error{#1}}
231 \def\bbbl@warning#1{%
232   \begingroup
233     \def\{\MessageBreak}%
234     \PackageWarning{babel}{#1}%
235   \endgroup}
236 \def\bbbl@infowarn#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \PackageNote{babel}{#1}%
240   \endgroup}
241 \def\bbbl@info#1{%
242   \begingroup
243     \def\{\MessageBreak}%
244     \PackageInfo{babel}{#1}%

```

```
245 \endgroup}
```

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

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

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

```
246 <<Basic macros>>
247 \ifpackagewith{babel}{silent}
248 {\let\bb@info\@gobble
249 \let\bb@infowarn\@gobble
250 \let\bb@warning\@gobble}
251 {}
252 %
253 \def\AfterBabelLanguage#1{%
254 \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.

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

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

```
292 \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`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\@for` or load `keyval`, for example.

```
293 \bbl@trace{key=value and another general options}
294 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
295 \def\bbl@tempb#1.#2{% Remove trailing dot
296   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
297 \def\bbl@tempe#1=#2\@@{%
298   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
299 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
300   \ifx\@empty#2%
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302   \else
303     \in@{,provide=}{, #1}%
304     \ifin@
305       \edef\bbl@tempc{%
306         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
307     \else
308       \in@{ $modifiers$ }{ $#1$ }% TODO. Allow spaces.
309       \ifin@
310         \bbl@tempe#2\@@
311       \else
312         \in@{=}{ #1}%
313         \ifin@
314           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
315         \else
316           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
317           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
318         \fi
319       \fi
320     \fi
321   \fi}
322 \let\bbl@tempc\@empty
323 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
324 \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.

```
325 \DeclareOption{KeepShorthandsActive}{}
326 \DeclareOption{activeacute}{}
327 \DeclareOption{activegrave}{}
328 \DeclareOption{debug}{}
329 \DeclareOption{noconfigs}{}
330 \DeclareOption{showlanguages}{}
331 \DeclareOption{silent}{}
332 % \DeclareOption{mono}{}
333 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
334 \chardef\bbl@iniflag\z@
335 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
336 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
337 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
338 % A separate option
339 \let\bbl@autoload@options\@empty
340 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
341 % Don't use. Experimental. TODO.
342 \newif\ifbbl@single
343 \DeclareOption{selectors=off}{\bbl@singletrue}
```

```
344 <<More package options>>
```

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

```
345 \let\bbl@opt@shorthands\@nnil
346 \let\bbl@opt@config\@nnil
347 \let\bbl@opt@main\@nnil
348 \let\bbl@opt@headfoot\@nnil
349 \let\bbl@opt@layout\@nnil
350 \let\bbl@opt@provide\@nnil
```

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

```
351 \def\bbl@tempa#1=#2\bbl@tempa{%
352   \bbl@csarg\ifx{opt#1}\@nnil
353     \bbl@csarg\edef{opt#1}{#2}%
354   \else
355     \bbl@error{bad-package-option}{#1}{#2}{}%
356   \fi}
```

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

```
357 \let\bbl@language@opts\@empty
358 \DeclareOption*{%
359   \bbl@xin@{\string=}{\CurrentOption}%
360   \ifin@
361     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
362   \else
363     \bbl@add@list\bbl@language@opts{\CurrentOption}%
364   \fi}
```

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

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

### 3.5 Conditional loading of shorthands

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

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

```
378 \bbl@trace{Conditional loading of shorthands}
379 \def\bbl@sh@string#1{%
380   \ifx#1\@empty\else
381     \ifx#1t\string~%
382     \else\ifx#1c\string,%
383     \else\string#1%
384   \fi\fi
385   \expandafter\bbl@sh@string
386 \fi}
```

```

387 \ifx\bbbl@opt@shorthands\@nnil
388   \def\bbbl@ifshorthand#1#2#3{#2}%
389 \else\ifx\bbbl@opt@shorthands\@empty
390   \def\bbbl@ifshorthand#1#2#3{#3}%
391 \else

```

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

```

392   \def\bbbl@ifshorthand#1{%
393     \bbbl@xin@\string#1}{\bbbl@opt@shorthands}%
394     \ifin@
395     \expandafter\@firstoftwo
396     \else
397     \expandafter\@secondoftwo
398     \fi}

```

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

```

399   \edef\bbbl@opt@shorthands{%
400     \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

401   \bbbl@ifshorthand{'}%
402     {\PassOptionsToPackage{activeacute}{babel}}{}
403   \bbbl@ifshorthand{`}%
404     {\PassOptionsToPackage{activegrave}{babel}}{}
405 \fi\fi

```

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

```

406 \ifx\bbbl@opt@headfoot\@nnil\else
407   \g@addto@macro\@resetactivechars{%
408     \set@typeset@protect
409     \expandafter\select@language\x\expandafter{\bbbl@opt@headfoot}%
410     \let\protect\noexpand}
411 \fi

```

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

```

412 \ifx\bbbl@opt@safe\@undefined
413   \def\bbbl@opt@safe{BR}
414   % \let\bbbl@opt@safe\@empty % Pending of \cite
415 \fi

```

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

```

416 \bbbl@trace{Defining IfBabelLayout}
417 \ifx\bbbl@opt@layout\@nnil
418   \newcommand\IfBabelLayout[3]{#3}%
419 \else
420   \bbbl@exp{\bbbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
421     \in{, layout, },{, #1,}%
422     \ifin@
423       \def\bbbl@opt@layout{#2}%
424       \bbbl@replace\bbbl@opt@layout{ }{.}%
425       \fi}
426   \newcommand\IfBabelLayout[1]{%
427     \@expandtwoargs\in{.#1.}{.\bbbl@opt@layout.}%
428     \ifin@
429     \expandafter\@firstoftwo
430     \else
431     \expandafter\@secondoftwo
432     \fi}
433 \fi
434 </package>
435 <*core>

```

### 3.6 Interlude for Plain

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

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

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

```
444 </core>
445 <*package | core>
```

## 4 Multiple languages

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

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

```
446 \def\bbbl@version{\<version>}
447 \def\bbbl@date{\<date>}
448 <<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.

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

`\bbbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bbbl@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.

```
464 \def\bbbl@fixname#1{%
465   \begingroup
466     \def\bbbl@tempe{l@}%
467     \edef\bbbl@tempd{\noexpand\@ifundefined{\noexpand\bbbl@tempe#1}}%
468     \bbbl@tempd
469     {\lowercase\expandafter{\bbbl@tempd}}%
470     {\uppercase\expandafter{\bbbl@tempd}}%
471     \@empty
472     {\edef\bbbl@tempd{\def\noexpand#1{\#1}}%
473       \uppercase\expandafter{\bbbl@tempd}}}%

```

```

474         {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
475         \lowercase\expandafter{\bbl@tempd}}}%
476     \@empty
477     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
478     \bbl@tempd
479     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
480 \def\bbl@iflanguage#1{%
481     \ifundefined{l@#1}{\@nolannerr{#1}\@gobble}\@firstofone}

```

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

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

```

482 \def\bbl@bcpcase#1#2#3#4\@@#5{%
483     \ifx\@empty#3%
484         \uppercase{\def#5{#1#2}}%
485     \else
486         \uppercase{\def#5{#1}}%
487         \lowercase{\edef#5{#5#2#3#4}}%
488     \fi}
489 \def\bbl@bcpllookup#1-#2-#3-#4\@@{%
490     \let\bbl@bcp\relax
491     \lowercase{\def\bbl@tempa{#1}}%
492     \ifx\@empty#2%
493         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
494     \else\ifx\@empty#3%
495         \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
496         \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
497             {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
498             {}%
499         \ifx\bbl@bcp\relax
500             \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
501         \fi
502     \else
503         \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
504         \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
505         \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
506             {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
507             {}%
508         \ifx\bbl@bcp\relax
509             \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
510                 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
511                 {}%
512         \fi
513         \ifx\bbl@bcp\relax
514             \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
515                 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
516                 {}%
517         \fi
518         \ifx\bbl@bcp\relax
519             \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
520         \fi
521     \fi\fi}
522 \let\bbl@initoload\relax
523 <-core>
524 \def\bbl@provide@locale{%
525     \ifx\babelprovide\@undefined
526         \bbl@error{base-on-the-fly}{}}}%
527     \fi
528     \let\bbl@auxname\language % Still necessary. TODO
529     \bbl@ifunset{\bbl@bcp@map@language}{}% Move uplevel??
530     {\edef\language{\@nameuse{\bbl@bcp@map@language}}}%

```

```

531 \ifbbl@bcpallowed
532   \expandafter\ifx\csname date\language\endcsname\relax
533     \expandafter
534     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@
535     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
536       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
537       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
538       \expandafter\ifx\csname date\language\endcsname\relax
539         \let\bbl@initoload\bbl@bcp
540         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
541         \let\bbl@initoload\relax
542       \fi
543       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
544     \fi
545   \fi
546 \fi
547 \expandafter\ifx\csname date\language\endcsname\relax
548   \IfFileExists{babel-\language.tex}%
549   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
550   {}%
551 \fi}
552 (+core)

```

**\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.

```

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

```

## 4.1 Selecting the language

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

```

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

```

564 \ifx\@undefined\protect\let\protect\relax\fi

```

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

```

565 \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.

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

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

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

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

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

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

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

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

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

```

602     }%
603     \fi}%
604     }%
605     \chardef\localeid\bbl@cl{id@}}

```

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

```

606 \expandafter\def\csname selectlanguage \endcsname#1{%
607   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
608   \bbl@push@language
609   \aftergroup\bbl@pop@language
610   \bbl@set@language{#1}}
611 \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 `\language` are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining `\BabelContentsFiles`, but make sure they are loaded inside a group (as `aux`, `toc`, `lof`, and `lot` do) or the last language of the document will remain active afterwards.

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

```

612 \def\BabelContentsFiles{toc,lof,lot}
613 \def\bbl@set@language#1{% from selectlanguage, pop@
614   % The old buggy way. Preserved for compatibility.
615   \edef\language#1%
616   \ifnum\escapechar=\expandafter`\string#1\@empty
617     \else\string#1\@empty\fi}%
618   \ifcat\relax\noexpand#1%
619     \expandafter\ifx\csname date\language\endcsname\relax
620       \edef\language#1%
621       \let\localename\language
622     \else
623       \bbl@info{Using '\string\language' instead of 'language' is\\%
624         deprecated. If what you want is to use a\\%
625         macro containing the actual locale, make\\%
626         sure it does not not match any language.\\%
627         Reported}%
628       \ifx\scantokens\@undefined
629         \def\localename{??}%
630       \else
631         \scantokens\expandafter{\expandafter
632           \def\expandafter\localename\expandafter{\language}}%
633       \fi
634     \fi
635   \else
636     \def\localename#1% This one has the correct catcodes
637   \fi
638   \select@language{\language}%
639   % write to auxs
640   \expandafter\ifx\csname date\language\endcsname\relax\else
641     \if@filesw
642       \ifx\babel@aux\@gobbles\else % Set if single in the first, redundant
643         \bbl@savelastskip
644         \protected@write\@auxout{\string\babel@aux{\bbl@auxname}}{}%
645         \bbl@restorelastskip
646       \fi
647       \bbl@usehooks{write}{}%
648     \fi

```

```

649 \fi}
650 %
651 \let\bbl@restorelastskip\relax
652 \let\bbl@savelastskip\relax
653 %
654 \newif\ifbbl@bcpallowed
655 \bbl@bcpallowedfalse
656 \def\select@language#1{% from set@, babel@aux
657   \ifx\bbl@selectorname\@empty
658     \def\bbl@selectorname{select}%
659   % set hymap
660   \fi
661   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
662   % set name
663   \edef\language#1}%
664   \bbl@fixname\language
665   % TODO. name@map must be here?
666   \bbl@provide@locale
667   \bbl@iflanguage\language{%
668     \let\bbl@select@type\z@
669     \expandafter\bbl@switch\expandafter{\language}}
670 \def\babel@aux#1#2{%
671   \select@language{#1}%
672   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
673     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
674 \def\babel@toc#1#2{%
675   \select@language{#1}}

```

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

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

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

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

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

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

```

676 \newif\ifbbl@usedategroup
677 \let\bbl@savedextras\@empty
678 \def\bbl@switch#1{% from select@, foreign@
679   % make sure there is info for the language if so requested
680   \bbl@ensureinfo{#1}%
681   % restore
682   \originalTeX
683   \expandafter\def\expandafter\originalTeX\expandafter{%
684     \csname noextras#1\endcsname
685     \let\originalTeX\@empty
686     \babel@beginsave}%
687   \bbl@usehooks{afterreset}}}%
688   \languageshorthands{none}%
689   % set the locale id
690   \bbl@id@assign
691   % switch captions, date
692   \bbl@bsphack
693   \ifcase\bbl@select@type
694     \csname captions#1\endcsname\relax
695     \csname date#1\endcsname\relax
696   \else

```

```

697 \bbl@xin@{,captions,}{, \bbl@select@opts,}%
698 \ifin@
699 \csname captions#1\endcsname\relax
700 \fi
701 \bbl@xin@{,date,}{, \bbl@select@opts,}%
702 \ifin@ % if \foreign... within \<lang>date
703 \csname date#1\endcsname\relax
704 \fi
705 \fi
706 \bbl@esphack
707 % switch extras
708 \csname bbl@preextras@#1\endcsname
709 \bbl@usehooks{beforeextras}{}%
710 \csname extras#1\endcsname\relax
711 \bbl@usehooks{afterextras}{}%
712 % > babel-ensure
713 % > babel-sh-<short>
714 % > babel-bidi
715 % > babel-fontspec
716 \let\bbl@savextras\empty
717 % hyphenation - case mapping
718 \ifcase\bbl@opt@hyphenmap\or
719 \def\BabelLower##1##2{\lccode##1=##2\relax}%
720 \ifnum\bbl@hymapsel>4\else
721 \csname\language\name @bbl@hyphenmap\endcsname
722 \fi
723 \chardef\bbl@opt@hyphenmap\z@
724 \else
725 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
726 \csname\language\name @bbl@hyphenmap\endcsname
727 \fi
728 \fi
729 \let\bbl@hymapsel@cclv
730 % hyphenation - select rules
731 \ifnum\csname l@language\endcsname=\l@unhyphenated
732 \edef\bbl@tempa{u}%
733 \else
734 \edef\bbl@tempa{\bbl@cclv\lnbrk}%
735 \fi
736 % linebreaking - handle u, e, k (v in the future)
737 \bbl@xin@{/u}{/\bbl@tempa}%
738 \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
739 \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
740 \ifin@\else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
741 \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
742 \ifin@
743 % unhyphenated/kashida/elongated/padding = allow stretching
744 \language\l@unhyphenated
745 \babel@savevariable\emergencystretch
746 \emergencystretch\maxdimen
747 \babel@savevariable\hbadness
748 \hbadness\@M
749 \else
750 % other = select patterns
751 \bbl@patterns{#1}%
752 \fi
753 % hyphenation - mins
754 \babel@savevariable\lefthyphenmin
755 \babel@savevariable\righthyphenmin
756 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
757 \set@hyphenmins\tw@\thr@@\relax
758 \else
759 \expandafter\expandafter\expandafter\set@hyphenmins

```

```

760 \csname #1hyphenmins\endcsname\relax
761 \fi
762 % reset selector name
763 \let\bbl@selectorname\@empty}

```

`otherlanguage (env.)` The `otherlanguage` environment 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.

```

764 \long\def\otherlanguage#1{%
765 \def\bbl@selectorname{other}%
766 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
767 \csname selectlanguage \endcsname{#1}%
768 \ignorespaces}

```

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

```

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

```

`otherlanguage* (env.)` The `otherlanguage` environment is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as ‘figure’. This environment makes use of `\foreign@language`.

```

770 \expandafter\def\csname otherlanguage*\endcsname{%
771 \ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
772 \def\bbl@otherlanguage@s[#1]#2{%
773 \def\bbl@selectorname{other*}%
774 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
775 \def\bbl@select@opts{#1}%
776 \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”.

```

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

```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.

Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras<lang>` command doesn’t make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

(3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in `vmode` and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into `hmode` with the surrounding `lang`, and with `\foreignlanguage*` with the new `lang`.

```

778 \providecommand\bbl@beforeforeign{}
779 \edef\foreignlanguage{%
780 \noexpand\protect
781 \expandafter\noexpand\csname foreignlanguage \endcsname}
782 \expandafter\def\csname foreignlanguage \endcsname{%
783 \ifstar\bbl@foreign@s\bbl@foreign@x}
784 \providecommand\bbl@foreign@x[3][]{%
785 \beginngroup
786 \def\bbl@selectorname{foreign}%

```

```

787 \def\bbl@select@opts{#1}%
788 \let\BabelText\@firstofone
789 \bbl@beforeforeign
790 \foreign@language{#2}%
791 \bbl@usehooks{foreign}{}%
792 \BabelText{#3}% Now in horizontal mode!
793 \endgroup}
794 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
795 \begingroup
796 {\par}%
797 \def\bbl@selectorname{foreign*}%
798 \let\bbl@select@opts\@empty
799 \let\BabelText\@firstofone
800 \foreign@language{#1}%
801 \bbl@usehooks{foreign*}{}%
802 \bbl@dirparastext
803 \BabelText{#2}% Still in vertical mode!
804 {\par}%
805 \endgroup}

```

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

```

806 \def\foreign@language#1{%
807 % set name
808 \edef\language{#1}%
809 \ifbbl@usedategroup
810 \bbl@add\bbl@select@opts{,date,}%
811 \bbl@usedategroupfalse
812 \fi
813 \bbl@fixname\language
814 % TODO. name@map here?
815 \bbl@provide@locale
816 \bbl@iflanguage\language{%
817 \let\bbl@select@type\@ne
818 \expandafter\bbl@switch\expandafter{\language}}

```

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

```

819 \def\IfBabelSelectorTF#1{%
820 \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
821 \ifin@
822 \expandafter\@firstoftwo
823 \else
824 \expandafter\@secondoftwo
825 \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.

```

826 \let\bbl@hyphlist\@empty
827 \let\bbl@hyphenation@relax
828 \let\bbl@pttnlist\@empty
829 \let\bbl@patterns@relax
830 \let\bbl@hymapsel=\@cclv
831 \def\bbl@patterns#1{%
832 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
833 \csname l@#1\endcsname
834 \edef\bbl@tempa{#1}%

```

```

835 \else
836 \csname l@#1:f@encoding\endcsname
837 \edef\bbl@tempa{#1:f@encoding}%
838 \fi
839 \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
840 % > luatex
841 \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
842 \begingroup
843 \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
844 \ifin@else
845 \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
846 \hyphenation{%
847 \bbl@hyphenation@
848 \@ifundefined{bbl@hyphenation@#1}%
849 \empty
850 {\space\csname bbl@hyphenation@#1\endcsname}}%
851 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
852 \fi
853 \endgroup}}

```

hyphenrules (*env.*) The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change `\language` and when the hyphenation rules specified were not loaded it has no effect. Note however, `\lccode`'s and font encodings are not set at all, so in most cases you should use `otherlanguage*`.

```

854 \def\hyphenrules#1{%
855 \edef\bbl@tempf{#1}%
856 \bbl@fixname\bbl@tempf
857 \bbl@iflanguage\bbl@tempf{%
858 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
859 \ifx\languageshorthands@undefined\else
860 \languageshorthands{none}%
861 \fi
862 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
863 \set@hyphenmins\tw@\thr@@\relax
864 \else
865 \expandafter\expandafter\expandafter\set@hyphenmins
866 \csname\bbl@tempf hyphenmins\endcsname\relax
867 \fi}}
868 \let\endhyphenrules\empty

```

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\<lang>hyphenmins` is already defined this command has no effect.

```

869 \def\providehyphenmins#1#2{%
870 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
871 \namedef{#1hyphenmins}{#2}%
872 \fi}

```

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

```

873 \def\set@hyphenmins#1#2{%
874 \lefthyphenmin#1\relax
875 \righthyphenmin#2\relax}

```

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

```

876 \ifx\ProvidesFile@undefined
877 \def\ProvidesLanguage#1[#2 #3 #4]{%
878 \wlog{Language: #1 #4 #3 <#2>}%
879 }

```

```

880 \else
881   \def\ProvidesLanguage#1{%
882     \begingroup
883     \catcode\ 10 %
884     \@makeother\/%
885     \ifnextchar[%]
886       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
887   \def\@provideslanguage#1[#2]{%
888     \wlog{Language: #1 #2}%
889     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
890     \endgroup}
891 \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`.

```
892 \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.

```
893 \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’:

```

894 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
895 \let\uselocale\setlocale
896 \let\locale\setlocale
897 \let\selectlocale\setlocale
898 \let\textlocale\setlocale
899 \let\textlanguage\setlocale
900 \let\languagegetext\setlocale

```

## 4.2 Errors

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

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.  
When the format knows about `\PackageError` it must be  $\TeX 2\epsilon$ , 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.

```

901 \edef\bbl@nulllanguage{\string\language=0}
902 \def\bbl@nocaption{\protect\bbl@nocaption@i}
903 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
904   \global\@namedef{#2}{\textbf{?#1?}}%
905   \@nameuse{#2}%
906   \edef\bbl@tempa{#1}%
907   \bbl@sreplace\bbl@tempa{name}}}%
908 \bbl@warning{%
909   \@backslashchar#1 not set for '\language'. Please,\\%
910   define it after the language has been loaded\\%
911   (typically in the preamble) with:\\%
912   \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
913   Feel free to contribute on github.com/latex3/babel.\\%
914   Reported}}
915 \def\bbl@tentative{\protect\bbl@tentative@i}
916 \def\bbl@tentative@i#1{%
917   \bbl@warning{%
918     Some functions for '#1' are tentative.\\%
919     They might not work as expected and their behavior\\%
920     could change in the future.\\%

```



```

921     Reported}}
922 \def\nolanerr#1{\bbl@error{undefined-language}{#1}{}}
923 \def\nopatterns#1{%
924   \bbl@warning
925     {No hyphenation patterns were preloaded for\%
926     the language '#1' into the format.\%
927     Please, configure your TeX system to add them and\%
928     rebuild the format. Now I will use the patterns\%
929     preloaded for \bbl@nulllanguage\space instead}}
930 \let\bbl@usehooks\@gobbletwo
931 \ifx\bbl@onlyswitch\@empty\endinput\fi
932 % Here ended switch.def

```

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

```

933 \ifx\directlua\@undefined\else
934   \ifx\bbl@luapatterns\@undefined
935     \input luababel.def
936   \fi
937 \fi
938 \bbl@trace{Compatibility with language.def}
939 \ifx\bbl@languages\@undefined
940   \ifx\directlua\@undefined
941     \openin1 = language.def % TODO. Remove hardcoded number
942     \ifeof1
943       \closein1
944       \message{I couldn't find the file language.def}
945     \else
946       \closein1
947       \begingroup
948         \def\addlanguage#1#2#3#4#5{%
949           \expandafter\ifx\csname lang@#1\endcsname\relax\else
950             \global\expandafter\let\csname l@#1\expandafter\endcsname
951             \csname lang@#1\endcsname
952           \fi}%
953         \def\uselanguage#1{%
954           \input language.def
955         \endgroup
956       \fi
957     \fi
958   \chardef\l@english\z@
959 \fi

```

`\addto` It takes two arguments, a *<control sequence>* and T<sub>E</sub>X-code to be added to the *<control sequence>*. If the *<control sequence>* has not been defined before it is defined now. The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```

960 \def\addto#1#2{%
961   \ifx#1\@undefined
962     \def#1{#2}%
963   \else
964     \ifx#1\relax
965       \def#1{#2}%
966     \else
967       {\toks@\expandafter{#1#2}%
968        \xdef#1{the\toks@}}%
969     \fi
970   \fi}

```

The macro `\initiate@active@char` below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool.

```

971 \def\bbl@withactive#1#2{%
972   \begingroup

```

```

973 \lccode`~=#2\relax
974 \lowercase{\endgroup#1~}}

```

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

```

975 \def\bbl@redefine#1{%
976 \edef\bbl@tempa{\bbl@stripslash#1}%
977 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
978 \expandafter\def\csname\bbl@tempa\endcsname}
979 \@onlypreamble\bbl@redefine

```

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

```

980 \def\bbl@redefine@long#1{%
981 \edef\bbl@tempa{\bbl@stripslash#1}%
982 \expandafter\let\csname org@\bbl@tempa\endcsname#1%
983 \long\expandafter\def\csname\bbl@tempa\endcsname}
984 \@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_`.

```

985 \def\bbl@redefineroobust#1{%
986 \edef\bbl@tempa{\bbl@stripslash#1}%
987 \bbl@ifunset{\bbl@tempa\space}%
988 {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
989 \bbl@exp{\def\#1{\protect<\bbl@tempa\space>}}}%
990 {\bbl@exp{\let<org@\bbl@tempa><\bbl@tempa\space>}}%
991 \namedef{\bbl@tempa\space}}
992 \@onlypreamble\bbl@redefineroobust

```

### 4.3 Hooks

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

```

993 \bbl@trace{Hooks}
994 \newcommand\AddBabelHook[3][[]]{%
995 \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
996 \def\bbl@tempa##1,#3=##2,##3\empty{\def\bbl@tempb{##2}}%
997 \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
998 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
999 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1000 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1001 \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1002 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1003 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1004 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1005 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1006 \ifx\UseHook\undefined\else\UseHook{babel/*/#2}\fi
1007 \def\bbl@elth##1{%
1008 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}}%
1009 \bbl@cs{ev@#2@#3}%
1010 \ifx\language\undefined\else % Test required for Plain (?)
1011 \ifx\UseHook\undefined\else\UseHook{babel/#1/#2}\fi
1012 \def\bbl@elth##1{%
1013 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}}%
1014 \bbl@cs{ev@#2@#1}%
1015 \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).

```

1016 \def\bbl@evargs{,% <- don't delete this comma
1017   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1018   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1019   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1020   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1021   beforestart=0,language=2,begindocument=1}
1022 \ifx\NewHook\undefined\else % Test for Plain (?)
1023   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1024   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1025 \fi

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@{language}`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro `\bbl@e@{language}` contains `\bbl@ensure{(include)}{(exclude)}{(fontenc)}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1026 \bbl@trace{Defining babelensure}
1027 \newcommand\babelensure[2][{}]{%
1028   \AddBabelHook{babel-ensure}{afterextras}{%
1029     \ifcase\bbl@select@type
1030       \bbl@cl{e}%
1031     \fi}%
1032   \begingroup
1033     \let\bbl@ens@include\empty
1034     \let\bbl@ens@exclude\empty
1035     \def\bbl@ens@fontenc{\relax}%
1036     \def\bbl@tempb##1{%
1037       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1038     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1039     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1040     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1041     \def\bbl@tempc{\bbl@ensure}%
1042     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1043       \expandafter{\bbl@ens@include}}%
1044     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1045       \expandafter{\bbl@ens@exclude}}%
1046     \toks@{\expandafter{\bbl@tempc}%
1047       \bbl@exp}%
1048   \endgroup
1049   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1050 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1051   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1052     \ifx##1\undefined % 3.32 - Don't assume the macro exists
1053       \edef##1{\noexpand\bbl@nocaption
1054         {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1055     \fi
1056     \ifx##1\@empty\else
1057       \in@{##1}{#2}%
1058     \fin@else
1059       \bbl@ifunset{\bbl@ensure@\language}%
1060       {\bbl@exp}%
1061       \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
1062         \\\foreignlanguage{\language}%
1063         {\ifx\relax#3\else
1064           \\\fontencoding{#3}\selectfont
1065         \fi

```

```

1066         #####1}}}%
1067     {}%
1068     \toks@\expandafter{##1}%
1069     \edef##1{%
1070         \bbl@csarg\noexpand{ensure@\language}%
1071         {\the\toks@}}%
1072     \fi
1073     \expandafter\bbl@tempb
1074     \fi}%
1075     \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1076     \def\bbl@tempa##1{% elt for include list
1077         \ifx##1\@empty\else
1078             \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1079             \ifin\@else
1080                 \bbl@tempb##1\@empty
1081             \fi
1082             \expandafter\bbl@tempa
1083         \fi}%
1084     \bbl@tempa#1\@empty}
1085 \def\bbl@captionslist{%
1086     \prefacename\refname\abstractname\bibname\chaptername\appendixname
1087     \contentsname\listfigurename\listtablename\indexname\figurename
1088     \tablename\partname\enclname\ccname\headtoname\pagename\seename
1089     \alsoname\proofname\glossaryname}

```

#### 4.4 Setting up language files

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

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

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

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

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

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

Finally we check `\originalTeX`.

```

1090 \bbl@trace{Macros for setting language files up}
1091 \def\bbl@ldfinit{%
1092     \let\bbl@screset\@empty
1093     \let\BabelStrings\bbl@opt@string
1094     \let\BabelOptions\@empty
1095     \let\BabelLanguages\relax
1096     \ifx\originalTeX\@undefined
1097         \let\originalTeX\@empty
1098     \else
1099         \originalTeX
1100     \fi}
1101 \def\LdfInit#1#2{%
1102     \chardef\atcatcode=\catcode` \@
1103     \catcode` \@=11\relax
1104     \chardef\eqcatcode=\catcode` \=
1105     \catcode` \=12\relax
1106     \expandafter\if\expandafter\@backslashchar
1107         \expandafter\@car\string#2\@nil

```

```

1108 \ifx#2\undefined\else
1109 \ldf@quit{#1}%
1110 \fi
1111 \else
1112 \expandafter\ifx\csname#2\endcsname\relax\else
1113 \ldf@quit{#1}%
1114 \fi
1115 \fi
1116 \bbl@ldfinit}

```

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

```

1117 \def\ldf@quit#1{%
1118 \expandafter\main@language\expandafter{#1}%
1119 \catcode\@=\atcatcode \let\atcatcode\relax
1120 \catcode\==\eqcatcode \let\eqcatcode\relax
1121 \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.

```

1122 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1123 \bbl@afterlang
1124 \let\bbl@afterlang\relax
1125 \let\BabelModifiers\relax
1126 \let\bbl@screset\relax}%
1127 \def\ldf@finish#1{%
1128 \loadlocalcfg{#1}%
1129 \bbl@afterldf{#1}%
1130 \expandafter\main@language\expandafter{#1}%
1131 \catcode\@=\atcatcode \let\atcatcode\relax
1132 \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 *LT<sub>ε</sub>X*.

```

1133 \@onlypreamble\LdfInit
1134 \@onlypreamble\ldf@quit
1135 \@onlypreamble\ldf@finish

```

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

```

1136 \def\main@language#1{%
1137 \def\bbl@main@language{#1}%
1138 \let\language\name\bbl@main@language % TODO. Set localename
1139 \bbl@id@assign
1140 \bbl@patterns{\language\name}}

```

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`.

```

1141 \def\bbl@beforestart{%
1142 \def\@nolanerr##1{%
1143 \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1144 \bbl@usehooks{beforestart}}}%
1145 \global\let\bbl@beforestart\relax}
1146 \AtBeginDocument{%
1147 {\@nameuse{bbl@beforestart}}% Group!
1148 \if@filesw
1149 \providecommand\babel@aux[2]{}%
1150 \immediate\write\@mainaux{%
1151 \string\providecommand\string\babel@aux[2]{}%

```

```

1152 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1153 \fi
1154 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1155 <-core>
1156 \ifx\bbl@normalsf\@empty
1157 \ifnum\sfcodes\@.=\@m
1158 \let\normalsfcodes\frenchspacing
1159 \else
1160 \let\normalsfcodes\nonfrenchspacing
1161 \fi
1162 \else
1163 \let\normalsfcodes\bbl@normalsf
1164 \fi
1165 <+core>
1166 \ifbbl@single % must go after the line above.
1167 \renewcommand\selectlanguage[1]{}%
1168 \renewcommand\foreignlanguage[2]{#2}%
1169 \global\let\babel@aux\@gobbletwo % Also as flag
1170 \fi}
1171 <-core>
1172 \AddToHook{begindocument/before}{%
1173 \let\bbl@normalsf\normalsfcodes
1174 \let\normalsfcodes\relax} % Hack, to delay the setting
1175 <+core>
1176 \ifcase\bbl@engine\or
1177 \AtBeginDocument{\pagedir\bodydir} % TODO - a better place
1178 \fi

```

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

```

1179 \def\select@language@x#1{%
1180 \ifcase\bbl@select@type
1181 \bbl@ifsamestring\language@name{#1}{\select@language{#1}}%
1182 \else
1183 \select@language{#1}%
1184 \fi}

```

## 4.5 Shorthands

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

```

1185 \bbl@trace{Shorhands}
1186 \def\bbl@add@special#1{% 1:a macro like \, \?, etc.
1187 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1188 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1189 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1190 \begingroup
1191 \catcode`#1\active
1192 \nfss@catcodes
1193 \ifnum\catcode`#1=\active
1194 \endgroup
1195 \bbl@add\nfss@catcodes{\@makeother#1}%
1196 \else
1197 \endgroup
1198 \fi
1199 \fi}

```

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

```

1200 \def\bbl@remove@special#1{%
1201   \begingroup
1202     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1203       \else\noexpand##1\noexpand##2\fi}%
1204     \def\do{\x\do}%
1205     \def\@makeother{\x\@makeother}%
1206   \edef\x{\endgroup
1207     \def\noexpand\dospecials{\dospecials}%
1208     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1209       \def\noexpand\@sanitize{\@sanitize}%
1210     \fi}%
1211   \x}

```

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

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

```

1212 \def\bbl@active@def#1#2#3#4{%
1213   \@namedef{#3#1}{%
1214     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1215       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1216     \else
1217       \bbl@afterfi\csname#2@sh@#1\endcsname
1218     \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.

```

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

```

1225 \def\initiate@active@char#1{%
1226   \bbl@ifunset{active@char\string#1}%
1227   {\bbl@withactive
1228     {\expandafter\initiate@active@char\expandafter}#1\string#1#1}%
1229   {}}

```

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).

```

1230 \def\@initiate@active@char#1#2#3{%
1231   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1232   \ifx#1\@undefined

```

```

1233 \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1234 \else
1235 \bbl@csarg\let{oridef@#2}#1%
1236 \bbl@csarg\edef{oridef@#2}{%
1237 \let\noexpand#1%
1238 \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1239 \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*").

```

1240 \ifx#1#3\relax
1241 \expandafter\let\csname normal@char#2\endcsname#3%
1242 \else
1243 \bbl@info{Making #2 an active character}%
1244 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1245 \@namedef{normal@char#2}{%
1246 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1247 \else
1248 \@namedef{normal@char#2}{#3}%
1249 \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).

```

1250 \bbl@restoreactive{#2}%
1251 \AtBeginDocument{%
1252 \catcode`#2\active
1253 \if@filesw
1254 \immediate\write\@mainaux{\catcode`\string#2\active}%
1255 \fi}%
1256 \expandafter\bbl@add@special\csname#2\endcsname
1257 \catcode`#2\active
1258 \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⟩`).

```

1259 \let\bbl@tempa\@firstoftwo
1260 \if\string^#2%
1261 \def\bbl@tempa{\noexpand\textormath}%
1262 \else
1263 \ifx\bbl@mathnormal\@undefined\else
1264 \let\bbl@tempa\bbl@mathnormal
1265 \fi
1266 \fi
1267 \expandafter\edef\csname active@char#2\endcsname{%
1268 \bbl@tempa
1269 {\noexpand\if@safe@actives
1270 \noexpand\expandafter
1271 \expandafter\noexpand\csname normal@char#2\endcsname
1272 \noexpand\else
1273 \noexpand\expandafter
1274 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1275 \noexpand\fi}%
1276 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1277 \bbl@csarg\edef{doactive#2}{%

```



```
1278 \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

```
\active@prefix <char> \normal@char<char>
```

(where `\active@char<char>` is *one* control sequence!).

```
1279 \bbl@csarg\edef{active@#2}{%
1280 \noexpand\active@prefix\noexpand#1%
1281 \expandafter\noexpand\csname active@char#2\endcsname}%
1282 \bbl@csarg\edef{normal@#2}{%
1283 \noexpand\active@prefix\noexpand#1%
1284 \expandafter\noexpand\csname normal@char#2\endcsname}%
1285 \bbl@ncarg\let#1{\bbl@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.

```
1286 \bbl@active@def#2\user@group{user@active}{language@active}%
1287 \bbl@active@def#2\language@group{language@active}{system@active}%
1288 \bbl@active@def#2\system@group{system@active}{normal@char}%
```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as `'` ends up in a heading  $\TeX$  would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```
1289 \expandafter\edef\csname\user@group @sh#2@\endcsname
1290 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1291 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1292 {\expandafter\noexpand\csname user@active#2\endcsname}%
```

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (`'`) active we need to change `\pr@ms` 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.

```
1293 \if\string'#2%
1294 \let\prim@s\bbl@prim@s
1295 \let\active@math@prime#1%
1296 \fi
1297 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
1298 <<More package options>> ≡
1299 \DeclareOption{math=active}{}
1300 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1301 <</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*.

```
1302 \@ifpackagewith{babel}{KeepShorthandsActive}%
1303 {\let\bbl@restoreactive@gobble}%
1304 {\def\bbl@restoreactive#1{%
1305 \bbl@exp{%
1306 \\\AfterBabelLanguage\\\CurrentOption
1307 {\catcode`#1=\the\catcode`#1\relax}%
1308 \\\AtEndOfPackage
1309 {\catcode`#1=\the\catcode`#1\relax}}}%
1310 \AtEndOfPackage{\let\bbl@restoreactive@gobble}}
```

`\bbl@sh@select` This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of `\hyphenation`. This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either `\bbl@firstcs` or `\bbl@scndcs`. Hence two more arguments need to follow it.

```

1311 \def\bbl@sh@select#1#2{%
1312   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1313     \bbl@afterelse\bbl@scndcs
1314   \else
1315     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1316   \fi}

```

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

```

1317 \begingroup
1318 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1319 {\gdef\active@prefix#1{%
1320   \ifx\protect\@typeset@protect
1321   \else
1322     \ifx\protect\@unexpandable@protect
1323       \noexpand#1%
1324     \else
1325       \protect#1%
1326     \fi
1327   \expandafter\@gobble
1328   \fi}}
1329 {\gdef\active@prefix#1{%
1330   \ifincsname
1331     \string#1%
1332     \expandafter\@gobble
1333   \else
1334     \ifx\protect\@typeset@protect
1335     \else
1336       \ifx\protect\@unexpandable@protect
1337         \noexpand#1%
1338       \else
1339         \protect#1%
1340       \fi
1341     \expandafter\expandafter\expandafter\@gobble
1342     \fi
1343   \fi}}
1344 \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`).

```

1345 \newif\if@safe@actives
1346 \@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.

```

1347 \def\bbl@restore@actives{\if@safe@actives\@safe@activefalse\fi}

```

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

```

1348 \chardef\bbl@activated\z@
1349 \def\bbl@activate#1{%
1350   \chardef\bbl@activated\@ne
1351   \bbl@withactive{\expandafter\let\expandafter}#1%
1352     \csname bbl@active@\string#1\endcsname}
1353 \def\bbl@deactivate#1{%
1354   \chardef\bbl@activated\tw@
1355   \bbl@withactive{\expandafter\let\expandafter}#1%
1356     \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs 1357 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1358 \def\bbl@scndcs#1#2{\csname#2\endcsname}

```

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

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

The auxiliary macro `\babel@texpdf` improves the interoperativity with `hyperref` and takes 4 arguments: (1) The  $\TeX$  code in text mode, (2) the string for `hyperref`, (3) the  $\TeX$  code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently `hyperref` doesn’t discriminate the mode). This macro may be used in `ldf` files.

```

1359 \def\babel@texpdf#1#2#3#4{%
1360   \ifx\texorpdfstring\@undefined
1361     \textormath{#1}{#3}%
1362   \else
1363     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1364     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1365   \fi}
1366 %
1367 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1368 \def\@decl@short#1#2#3\@nil#4{%
1369   \def\bbl@tempa{#3}%
1370   \ifx\bbl@tempa\@empty
1371     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1372     \bbl@ifunset{#1@sh@\string#2@}{}%
1373     {\def\bbl@tempa{#4}%
1374       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1375       \else
1376         \bbl@info
1377           {Redefining #1 shorthand \string#2\\%
1378            in language \CurrentOption}%
1379       \fi}%
1380     \@namedef{#1@sh@\string#2@}{#4}%
1381   \else
1382     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1383     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1384     {\def\bbl@tempa{#4}%
1385       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1386       \else
1387         \bbl@info
1388           {Redefining #1 shorthand \string#2\string#3\\%
1389            in language \CurrentOption}%
1390       \fi}%
1391     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1392   \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.

```
1393 \def\textormath{%
1394   \ifmmode
1395     \expandafter\@secondoftwo
1396   \else
1397     \expandafter\@firstoftwo
1398   \fi}
```

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

```
1399 \def\user@group{user}
1400 \def\language@group{english} % TODO. I don't like defaults
1401 \def\system@group{system}
```

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

```
1402 \def\useshorthands{%
1403   \ifstar\bb@usesh@s{\bb@usesh@x{}}
1404 \def\bb@usesh@s#1{%
1405   \bb@usesh@x
1406     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1407     {#1}}
1408 \def\bb@usesh@x#1#2{%
1409   \bb@ifshorthand{#2}%
1410   {\def\user@group{user}%
1411     \initiate@active@char{#2}%
1412     #1%
1413     \bb@activate{#2}}%
1414   {\bb@error{shorthand-is-off}{#2}{}}}
```

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

```
1415 \def\user@language@group{user@language@group}
1416 \def\bb@set@user@generic#1#2{%
1417   \bb@ifunset{user@generic@active#1}%
1418   {\bb@active@def#1\user@language@group{user@active}{user@generic@active}%
1419     \bb@active@def#1\user@group{user@generic@active}{language@active}%
1420     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1421       \expandafter\noexpand\csname normal@char#1\endcsname}%
1422     \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1423       \expandafter\noexpand\csname user@active#1\endcsname}}%
1424   \@empty}
1425 \newcommand\defineshorthand[3][user]{%
1426   \edef\bb@tempa{\zap@space#1 \@empty}%
1427   \bb@for\bb@tempb\bb@tempa{%
1428     \if*\expandafter\@car\bb@tempb\@nil
1429     \edef\bb@tempb{user@\expandafter\@gobble\bb@tempb}%
1430     \@expandtwoargs
1431     \bb@set@user@generic{\expandafter\string\@car#2\@nil}\bb@tempb
1432   \fi
1433   \declare@shorthand{\bb@tempb}{#2}{#3}}}
```

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

```
1434 \def\languageshorthands#1{\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`.

```

1435 \def\aliasshorthand#1#2{%
1436   \bbl@ifshorthand{#2}%
1437   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1438     \ifx\document\@notprerr
1439       \@notshorthand{#2}%
1440     \else
1441       \initiate@active@char{#2}%
1442       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1443       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1444       \bbl@activate{#2}%
1445     \fi
1446   \fi}%
1447   {\bbl@error{shorthand-is-off}{#2}{}}}
```

`\@notshorthand`

```

1448 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}
```

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

```

1449 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1450 \DeclareRobustCommand*\shorthandoff{%
1451   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1452 \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.

```

1453 \def\bbl@switch@sh#1#2{%
1454   \ifx#2\@nnil\else
1455     \bbl@ifunset{\bbl@active@\string#2}%
1456     {\bbl@error{not-a-shorthand-b}{#2}{}}%
1457     {\ifcase#1%   off, on, off*
1458       \catcode`#2\relax
1459     \or
1460       \catcode`#2\active
1461       \bbl@ifunset{\bbl@shdef@\string#2}%
1462       {}%
1463       {\bbl@withactive{\expandafter\let\expandafter}#2%
1464         \csname bbl@shdef@\string#2\endcsname
1465         \bbl@csarg\let{shdef@\string#2}\relax}%
1466       \ifcase\bbl@activated\or
1467         \bbl@activate{#2}%
1468       \else
1469         \bbl@deactivate{#2}%
1470       \fi
1471     \or
1472       \bbl@ifunset{\bbl@shdef@\string#2}%
1473       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2%
1474       {}%
1475       \csname bbl@oricat@\string#2\endcsname
1476       \csname bbl@oridef@\string#2\endcsname
1477       \fi}%
1478     \bbl@afterfi\bbl@switch@sh#1%
1479   \fi}
```

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

```

1480 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1481 \def\bbl@putsh#1{%
1482   \bbl@ifunset{bbl@active@\string#1}%
1483   {\bbl@putsh@i#1@empty\@nnil}%
1484   {\csname bbl@active@\string#1\endcsname}}
1485 \def\bbl@putsh@i#1#2\@nnil{%
1486   \csname\language@group @sh@\string#1@%
1487   \ifx\@empty#2\else\string#2@\fi\endcsname}
1488 %
1489 \ifx\bbl@opt@shorthands\@nnil\else
1490   \let\bbl@s@initiate@active@char\initiate@active@char
1491   \def\initiate@active@char#1{%
1492     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1493   \let\bbl@s@switch@sh\bbl@switch@sh
1494   \def\bbl@switch@sh#1#2{%
1495     \ifx#2\@nnil\else
1496       \bbl@afterfi
1497       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1498       \fi}
1499   \let\bbl@s@activate\bbl@activate
1500   \def\bbl@activate#1{%
1501     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1502   \let\bbl@s@deactivate\bbl@deactivate
1503   \def\bbl@deactivate#1{%
1504     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1505   \fi

```

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

```

1506 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}

```

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

```

1507 \def\bbl@prim@s{%
1508   \prime\futurelet\@let@token\bbl@pr@m@s}
1509 \def\bbl@if@primes#1#2{%
1510   \ifx#1\@let@token
1511     \expandafter\@firstoftwo
1512   \else\ifx#2\@let@token
1513     \bbl@afterelse\expandafter\@firstoftwo
1514   \else
1515     \bbl@afterfi\expandafter\@secondoftwo
1516   \fi\fi}
1517 \begingroup
1518 \catcode`\^=7 \catcode`\*=\active \lccode`\*=`^
1519 \catcode`\'=12 \catcode`\"=\active \lccode`\"=`'
1520 \lowercase{%
1521   \gdef\bbl@pr@m@s{%
1522     \bbl@if@primes" '%
1523     \pr@@s
1524     {\bbl@if@primes*^ \pr@@t\egroup}}}
1525 \endgroup

```

Usually the `~` is active and expands to `\penalty\@M\.`. When it is written to the `.aux` file it is written expanded. To prevent that and to be able to use the character `~` as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when `~` is still a non-break space), and in some cases is inconvenient (if `~` has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

```

1526 \initiate@active@char{~}
1527 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1528 \bbl@activate{~}

```

\OT1dqpos The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.

```

1529 \expandafter\def\csname OT1dqpos\endcsname{127}
1530 \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

```

1531 \ifx\f@encoding\@undefined
1532   \def\f@encoding{OT1}
1533 \fi

```

## 4.6 Language attributes

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

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

```

1534 \bbl@trace{Language attributes}
1535 \newcommand\languageattribute[2]{%
1536   \def\bbl@tempc{#1}%
1537   \bbl@fixname\bbl@tempc
1538   \bbl@iflanguage\bbl@tempc{%
1539     \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.

```

1540     \ifx\bbl@known@attrs\@undefined
1541       \in@false
1542     \else
1543       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1544     \fi
1545     \ifin@
1546       \bbl@warning{%
1547         You have more than once selected the attribute '##1'\%
1548         for language #1. Reported}%
1549     \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.

```

1550       \bbl@exp{%
1551         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1552       \edef\bbl@tempa{\bbl@tempc-##1}%
1553       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1554       {\csname\bbl@tempc @attr##1\endcsname}%
1555       {\@attrerr{\bbl@tempc}{##1}}%
1556     \fi}}%
1557 \@onlypreamble\languageattribute

```

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

```

1558 \newcommand*{\@attrerr}[2]{%
1559   \bbl@error{unknown-attribute}{#1}{#2}{}}

```

\bbl@declare@attribute 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}.

```

1560 \def\bbl@declare@ttribute#1#2#3{%
1561   \bbl@xin@{,#2,},{,\BabelModifiers,}%
1562   \ifin@
1563     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1564   \fi
1565   \bbl@add@list\bbl@attributes{#1-#2}%
1566   \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.

```

1567 \def\bbl@ifattributeset#1#2#3#4{%
1568   \ifx\bbl@known@attrs\@undefined
1569     \in@false
1570   \else
1571     \bbl@xin@{,#1-#2,},{,\bbl@known@attrs,}%
1572   \fi
1573   \ifin@
1574     \bbl@afterelse#3%
1575   \else
1576     \bbl@afterfi#4%
1577   \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.

```

1578 \def\bbl@ifknown@ttrib#1#2{%
1579   \let\bbl@tempa\@secondoftwo
1580   \bbl@loopx\bbl@tempb{#2}{%
1581     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1582     \ifin@
1583       \let\bbl@tempa\@firstoftwo
1584     \else
1585       \fi}%
1586   \bbl@tempa}

```

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

```

1587 \def\bbl@clear@ttribs{%
1588   \ifx\bbl@attributes\@undefined\else
1589     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1590       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1591     \let\bbl@attributes\@undefined
1592   \fi}
1593 \def\bbl@clear@ttrib#1-#2.{%
1594   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1595 \AtBeginDocument{\bbl@clear@ttribs}

```

## 4.7 Support for saving macro definitions

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



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

`\babel@beginsave` 1596 `\bbl@trace{Macros for saving definitions}`  
1597 `\def\babel@beginsave{\babel@savecnt\z@}`

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

1598 `\newcount\babel@savecnt`

1599 `\babel@beginsave`

`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX`<sup>2</sup>. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable⟨variable⟩` saves the value of the variable. `⟨variable⟩` can be anything allowed after the `\the` primitive. To avoid messing saved definitions up, they are saved only the very first time.

```
1600 \def\babel@save#1{%
1601   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1602   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1603     \expandafter{\expandafter,\bbl@savextras,}}%
1604   \expandafter\in@\bbl@tempa
1605   \ifin@ \else
1606     \bbl@add\bbl@savextras{,{#1,}}%
1607     \bbl@carg\let{babel@\number\babel@savecnt}#1\relax
1608     \toks@\expandafter{\originalTeX\let#1=}
1609     \bbl@exp{%
1610       \def\\originalTeX{\the\toks@<babel@\number\babel@savecnt>\relax}}%
1611     \advance\babel@savecnt@ne
1612   \fi}
1613 \def\babel@savevariable#1{%
1614   \toks@\expandafter{\originalTeX #1=}
1615   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@nonfrenchspacing` switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in `\babelprovide`. This new method should be ideally the default one.

```
1616 \def\bbl@frenchspacing{%
1617   \ifnum\the\sfcode`.=\@m
1618     \let\bbl@nonfrenchspacing\relax
1619   \else
1620     \frenchspacing
1621     \let\bbl@nonfrenchspacing\nonfrenchspacing
1622   \fi}
1623 \let\bbl@nonfrenchspacing\nonfrenchspacing
1624 \let\bbl@elt\relax
1625 \edef\bbl@fs@chars{%
1626   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1627   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1628   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1629 \def\bbl@pre@fs{%
1630   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##2\relax}%
1631   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1632 \def\bbl@post@fs{%
1633   \bbl@save@sfcodes
1634   \edef\bbl@tempa{\bbl@cl{frspc}}%
1635   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1636   \if u\bbl@tempa % do nothing
1637   \else\if n\bbl@tempa % non french
1638     \def\bbl@elt##1##2##3{%
1639       \ifnum\sfcode`##1=##2\relax
1640       \babel@savevariable{\sfcode`##1}%

```

<sup>2</sup>`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```

1641     \sfcode`##1=##3\relax
1642   \fi}%
1643   \bbl@fs@chars
1644 \else\if y\bbl@tempa    % french
1645   \def\bbl@elt##1##2##3{%
1646     \ifnum\sfcode`##1=##3\relax
1647       \babel@savevariable{\sfcode`##1}%
1648       \sfcode`##1=##2\relax
1649     \fi}%
1650   \bbl@fs@chars
1651   \fi\fi\fi}

```

## 4.8 Short tags

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

```

1652 \bbl@trace{Short tags}
1653 \def\babeltags#1{%
1654   \edef\bbl@tempa{\zap@space#1 \@empty}%
1655   \def\bbl@tempb##1=##2\@{ }%
1656   \edef\bbl@tempc{%
1657     \noexpand\newcommand
1658     \expandafter\noexpand\csname ##1\endcsname{%
1659       \noexpand\protect
1660       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1661     \noexpand\newcommand
1662     \expandafter\noexpand\csname text##1\endcsname{%
1663       \noexpand\foreignlanguage{##2}}
1664   \bbl@tempc}%
1665   \bbl@for\bbl@tempa\bbl@tempa{%
1666     \expandafter\bbl@tempb\bbl@tempa\@{ }

```

## 4.9 Hyphens

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

```

1667 \bbl@trace{Hyphens}
1668 \@onlypreamble\babelhyphenation
1669 \AtEndOfPackage{%
1670   \newcommand\babelhyphenation[2][\@empty]{%
1671     \ifx\bbl@hyphenation@\relax
1672       \let\bbl@hyphenation@\@empty
1673     \fi
1674     \ifx\bbl@hyphlist\@empty\else
1675       \bbl@warning{%
1676         You must not intermingle \string\selectlanguage\space and\\%
1677         \string\babelhyphenation\space or some exceptions will not\\%
1678         be taken into account. Reported}%
1679       \fi
1680     \ifx\@empty#1%
1681       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1682     \else
1683       \bbl@vforeach{#1}{%
1684         \def\bbl@tempa{##1}%
1685         \bbl@fixname\bbl@tempa
1686         \bbl@iflanguage\bbl@tempa{%
1687           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1688             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1689             {}%
1690             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%

```

```

1691          #2}}}%
1692      \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`<sup>3</sup>.

```

1693 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1694 \def\bbl@t@one{Tl}
1695 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

`\babelhyphen` Macros to insert common hyphens. Note the space before `@` in `\babelhyphen`. Instead of protecting it with `\DeclareRobustCommand`, which could insert a `\relax`, we use the same procedure as shorthands, with `\active@prefix`.

```

1696 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1697 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1698 \def\bbl@hyphen{%
1699   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1700 \def\bbl@hyphen@i#1#2{%
1701   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
1702   {\csname bbl@#1usehyphen\endcsname\discretionary{#2}{#{#2}}}%
1703   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

```

1704 \def\bbl@usehyphen#1{%
1705   \leavevmode
1706   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1707   \nobreak\hskip\z@skip}
1708 \def\bbl@usehyphen#1{%
1709   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1710 \def\bbl@hyphenchar{%
1711   \ifnum\hyphenchar\font=\m@ne
1712     \babelnullhyphen
1713   \else
1714     \char\hyphenchar\font
1715   \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.

```

1716 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1717 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1718 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1719 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1720 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1721 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1722 \def\bbl@hy@repeat{%
1723   \bbl@usehyphen{%
1724     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1725 \def\bbl@hy@@repeat{%
1726   \bbl@usehyphen{%
1727     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1728 \def\bbl@hy@empty{\hskip\z@skip}
1729 \def\bbl@hy@@empty{\discretionary{}{}{}}

```

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

```

1730 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#{#1}}\bbl@allowhyphens}

```

<sup>3</sup>TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

## 4.10 Multiencoding strings

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

**Tools** But first, a tool. It makes global a local variable. This is not the best solution, but it works.

```
1731 \bbl@trace{Multiencoding strings}
1732 \def\bbl@tglobal#1{\global\let#1#1}
```

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

```
1733 <<{*More package options}>> ≡
1734 \DeclareOption{nocase}{}
1735 <</More package options>>
```

The following package options control the behavior of \SetString.

```
1736 <<{*More package options}>> ≡
1737 \let\bbl@opt@strings\@nnil % accept strings=value
1738 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1739 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1740 \def\BabelStringsDefault{generic}
1741 <</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.

```
1742 \@onlypreamble\StartBabelCommands
1743 \def\StartBabelCommands{%
1744   \begingroup
1745   \@tempcnta="7F
1746   \def\bbl@tempa{%
1747     \ifnum\@tempcnta>"FF\else
1748       \catcode\@tempcnta=11
1749       \advance\@tempcnta\@ne
1750       \expandafter\bbl@tempa
1751     \fi}%
1752   \bbl@tempa
1753   <<Macros local to BabelCommands>>
1754   \def\bbl@provstring##1##2{%
1755     \providecommand##1{##2}%
1756     \bbl@tglobal##1}%
1757   \global\let\bbl@scafter\@empty
1758   \let\StartBabelCommands\bbl@startcmds
1759   \ifx\BabelLanguages\relax
1760     \let\BabelLanguages\CurrentOption
1761   \fi
1762   \begingroup
1763   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1764   \StartBabelCommands}
1765 \def\bbl@startcmds{%
1766   \ifx\bbl@screset\@nnil\else
1767     \bbl@usehooks{stopcommands}{}%
1768   \fi
1769   \endgroup
1770   \begingroup
1771   \@ifstar
1772     {\ifx\bbl@opt@strings\@nnil
1773       \let\bbl@opt@strings\BabelStringsDefault
1774     \fi
1775     \bbl@startcmds@i}%
1776   \bbl@startcmds@i}
1777 \def\bbl@startcmds@i#1#2{%
1778   \edef\bbl@L{\zap@space#1 \@empty}}%
```

```

1779 \edef\bbI@G{\zap@space#2 \@empty}%
1780 \bbI@startcmds@ii}
1781 \let\bbI@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1782 \newcommand\bbI@startcmds@ii[1][\@empty]{%
1783 \let\SetString\@gobbletwo
1784 \let\bbI@stringdef\@gobbletwo
1785 \let\AfterBabelCommands\@gobble
1786 \ifx\@empty#1%
1787 \def\bbI@sc@label{generic}%
1788 \def\bbI@encstring##1##2{%
1789 \ProvideTextCommandDefault##1{##2}%
1790 \bbI@toglobal##1%
1791 \expandafter\bbI@toglobal\csname\string? \string##1\endcsname}%
1792 \let\bbI@sctest\in@true
1793 \else
1794 \let\bbI@sc@charset\space % <- zapped below
1795 \let\bbI@sc@fontenc\space % <- " "
1796 \def\bbI@tempa##1=##2\@nil{%
1797 \bbI@csarg\edef{sc@ \zap@space##1 \@empty}{##2 }}%
1798 \bbI@foreach{label=#1}{\bbI@tempa##1\@nil}%
1799 \def\bbI@tempa##1 ##2{% space -> comma
1800 ##1%
1801 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbI@afterfi\bbI@tempa##2\fi}%
1802 \edef\bbI@sc@fontenc{\expandafter\bbI@tempa\bbI@sc@fontenc\@empty}%
1803 \edef\bbI@sc@label{\expandafter\zap@space\bbI@sc@label\@empty}%
1804 \edef\bbI@sc@charset{\expandafter\zap@space\bbI@sc@charset\@empty}%
1805 \def\bbI@encstring##1##2{%
1806 \bbI@foreach\bbI@sc@fontenc{%
1807 \bbI@ifunset{T@###1}%
1808 }%
1809 {\ProvideTextCommand##1{###1}{##2}%
1810 \bbI@toglobal##1%
1811 \expandafter
1812 \bbI@toglobal\csname###1\string##1\endcsname}}}%
1813 \def\bbI@sctest{%
1814 \bbI@xin@{\bbI@opt@strings,}{,\bbI@sc@label,\bbI@sc@fontenc,}}%
1815 \fi
1816 \ifx\bbI@opt@strings\@nnil % ie, no strings key -> defaults
1817 \else\ifx\bbI@opt@strings\relax % ie, strings=encoded
1818 \let\AfterBabelCommands\bbI@aftercmds
1819 \let\SetString\bbI@setstring
1820 \let\bbI@stringdef\bbI@encstring
1821 \else % ie, strings=value
1822 \bbI@sctest
1823 \ifin@
1824 \let\AfterBabelCommands\bbI@aftercmds
1825 \let\SetString\bbI@setstring
1826 \let\bbI@stringdef\bbI@provstring
1827 \fi\fi\fi
1828 \bbI@scswitch
1829 \ifx\bbI@G\@empty
1830 \def\SetString##1##2{%
1831 \bbI@error{missing-group}{##1}{}}}%

```

```

1832 \fi
1833 \ifx\@empty#1%
1834 \bbl@usehooks{defaultcommands}{}%
1835 \else
1836 \@expandtwoargs
1837 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1838 \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).

```

1839 \def\bbl@forlang#1#2{%
1840 \bbl@for#1\bbl@L{%
1841 \bbl@xin@{, #1, }{, \BabelLanguages, }%
1842 \ifin@#2\relax\fi}}
1843 \def\bbl@scswitch{%
1844 \bbl@forlang\bbl@tempa{%
1845 \ifx\bbl@G\@empty\else
1846 \ifx\SetString\gobbletwo\else
1847 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1848 \bbl@xin@{, \bbl@GL, }{, \bbl@screset, }%
1849 \ifin@\else
1850 \global\expandafter\let\csname\bbl@GL\endcsname\undefined
1851 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1852 \fi
1853 \fi
1854 \fi}}
1855 \AtEndOfPackage{%
1856 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1857 \let\bbl@scswitch\relax}
1858 \onlypreamble\EndBabelCommands
1859 \def\EndBabelCommands{%
1860 \bbl@usehooks{stopcommands}{}%
1861 \endgroup
1862 \endgroup
1863 \bbl@scafter}
1864 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside `\StartBabelCommands`.

**Strings** The following macro is the actual definition of `\SetString` when it is “active”. First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like `\providescommand`). With the event `stringprocess` you can preprocess the string by manipulating the value of `\BabelString`. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

1865 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1866 \bbl@forlang\bbl@tempa{%
1867 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1868 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1869 {\bbl@exp{%
1870 \global\\bbl@add\<\bbl@G\bbl@tempa>{\bbl@scset\\#1\<\bbl@LC>}}}%
1871 }%
1872 \def\BabelString{#2}%
1873 \bbl@usehooks{stringprocess}{}%
1874 \expandafter\bbl@stringdef
1875 \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```
1876 \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.

```
1877 <<(*Macros local to BabelCommands)>> ≡
1878 \def\SetStringLoop##1##2{%
1879   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1880   \count@\z@
1881   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1882     \advance\count@\@ne
1883     \toks@\expandafter{\bbl@tempa}%
1884     \bbl@exp{%
1885       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1886       \count@=\the\count@\relax}}}%
1887 <</Macros local to BabelCommands>>
```

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

```
1888 \def\bbl@aftercmds#1{%
1889   \toks@\expandafter{\bbl@scafter#1}%
1890   \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.

```
1891 <<(*Macros local to BabelCommands)>> ≡
1892 \newcommand\SetCase[3][]{%
1893   \def\bbl@tempa####1####2{%
1894     \ifx####1\@empty\else
1895       \bbl@carg\bbl@add{extras\CurrentOption}{%
1896         \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1897         \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1898         \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1899         \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1900       \expandafter\bbl@tempa
1901     \fi}%
1902   \bbl@tempa##1\@empty\@empty
1903   \bbl@carg\bbl@toglobal{extras\CurrentOption}}%
1904 <</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.

```
1905 <<(*Macros local to BabelCommands)>> ≡
1906 \newcommand\SetHyphenMap[1]{%
1907   \bbl@forlang\bbl@tempa{%
1908     \expandafter\bbl@stringdef
1909     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1910 <</Macros local to BabelCommands>>
```

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

```
1911 \newcommand\BabelLower[2]{% one to one.
1912   \ifnum\lccode#1=#2\else
1913     \babel@savevariable{\lccode#1}%
1914     \lccode#1=#2\relax
1915   \fi}
1916 \newcommand\BabelLowerMM[4]{% many-to-many
1917   \@tempcnta=#1\relax
1918   \@tempcntb=#4\relax
1919   \def\bbl@tempa{%
1920     \ifnum\@tempcnta>#2\else
1921       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1922       \advance\@tempcnta#3\relax
```

```

1923 \advance\@tempcntb#3\relax
1924 \expandafter\bb\@tempa
1925 \fi}%
1926 \bb\@tempa}
1927 \newcommand\BabelLowerM0[4]{% many-to-one
1928 \@tempcnta=#1\relax
1929 \def\bb\@tempa{%
1930 \ifnum\@tempcnta>#2\else
1931 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1932 \advance\@tempcnta#3
1933 \expandafter\bb\@tempa
1934 \fi}%
1935 \bb\@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1936 <<*More package options>> ≡
1937 \DeclareOption{hyphenmap=off}{\chardef\bb\@opt@hyphenmap\z@}
1938 \DeclareOption{hyphenmap=first}{\chardef\bb\@opt@hyphenmap\@ne}
1939 \DeclareOption{hyphenmap=select}{\chardef\bb\@opt@hyphenmap\tw@}
1940 \DeclareOption{hyphenmap=other}{\chardef\bb\@opt@hyphenmap\thr@@}
1941 \DeclareOption{hyphenmap=other*}{\chardef\bb\@opt@hyphenmap4\relax}
1942 <</More package options>>

```

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

```

1943 \AtEndOfPackage{%
1944 \ifx\bb\@opt@hyphenmap\undefined
1945 \bb\@xin@{,}{\bb\@language@opts}%
1946 \chardef\bb\@opt@hyphenmap\ifin@4\else\@ne\fi
1947 \fi}

```

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```

1948 \newcommand\setlocalecaption{% TODO. Catch typos.
1949 \@ifstar\bb\@setcaption@{\bb\@setcaption@x}
1950 \def\bb\@setcaption@x#1#2#3{% language caption-name string
1951 \bb\@trim@def\bb\@tempa{#2}%
1952 \bb\@xin@{.template}{\bb\@tempa}%
1953 \ifin@
1954 \bb\@ini@captions@template{#3}{#1}%
1955 \else
1956 \edef\bb\@tempd{%
1957 \expandafter\expandafter\expandafter
1958 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1959 \bb\@xin@
1960 {\expandafter\string\csname #2name\endcsname}%
1961 {\bb\@tempd}%
1962 \ifin@ % Renew caption
1963 \bb\@xin@{\string\bb\@scset}{\bb\@tempd}%
1964 \ifin@
1965 \bb\@exp{%
1966 \\bb\@ifsamestring{\bb\@tempa}{\language}%
1967 {\bb\@scset\<#2name>\<#1#2name>}%
1968 {}}%
1969 \else % Old way converts to new way
1970 \bb\@ifunset{#1#2name}%
1971 {\bb\@exp{%
1972 \\bb\@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
1973 \\bb\@ifsamestring{\bb\@tempa}{\language}%
1974 {\def\<#2name>{\<#1#2name>}}}%
1975 {}}}%
1976 {}%
1977 \fi
1978 \else

```



```

1979 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1980 \ifin@ % New way
1981 \bbl@exp{%
1982 \\\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}}%
1983 \\\bbl@ifsamestring{\bbl@tempa}{\language\name}%
1984 {\bbl@scset\<#2name>\<#1#2name>}}%
1985 {}}%
1986 \else % Old way, but defined in the new way
1987 \bbl@exp{%
1988 \\\bbl@add\<captions#1>{\def\<#2name>\<#1#2name>}}%
1989 \\\bbl@ifsamestring{\bbl@tempa}{\language\name}%
1990 {\def\<#2name>\<#1#2name>}}%
1991 {}}%
1992 \fi%
1993 \fi
1994 \@namedef{#1#2name}{#3}%
1995 \toks@\expandafter{\bbl@captionslist}%
1996 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
1997 \ifin@ \else
1998 \bbl@exp{\bbl@add\<captionslist>\<#2name>}}%
1999 \bbl@toглобаль\bbl@captionslist
2000 \fi
2001 \fi}
2002 % \def\bbl@setcaption@#1#2#3{ % TODO. Not yet implemented (w/o 'name')

```

## 4.11 Macros common to a number of languages

`\set@low@box` The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```

2003 \bbl@trace{Macros related to glyphs}
2004 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2005 \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2006 \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.

```

2007 \def\save@sf@q#1{\leavevmode
2008 \begingroup
2009 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2010 \endgroup}

```

## 4.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through `Tlenc.def`.

### 4.12.1 Quotation marks

`\quotedblbase` In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```

2011 \ProvideTextCommand{\quotedblbase}{OT1}{%
2012 \save@sf@q{\set@low@box{\textquotedblright\}}%
2013 \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.

```

2014 \ProvideTextCommandDefault{\quotedblbase}{%
2015 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2016 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2017 \save@sf@q{\set@low@box{\textquoteright\}}%
2018 \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.

```
2019 \ProvideTextCommandDefault{\quotesinglbase}{%
2020   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2021 \ProvideTextCommand{\guillemetleft}{OT1}{%
2022   \ifmmode
2023     \ll
2024   \else
2025     \save@sf@q{\nobreak
2026       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2027   \fi}
2028 \ProvideTextCommand{\guillemetright}{OT1}{%
2029   \ifmmode
2030     \gg
2031   \else
2032     \save@sf@q{\nobreak
2033       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2034   \fi}
2035 \ProvideTextCommand{\guillemotleft}{OT1}{%
2036   \ifmmode
2037     \ll
2038   \else
2039     \save@sf@q{\nobreak
2040       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2041   \fi}
2042 \ProvideTextCommand{\guillemotright}{OT1}{%
2043   \ifmmode
2044     \gg
2045   \else
2046     \save@sf@q{\nobreak
2047       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2048   \fi}
```

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

```
2049 \ProvideTextCommandDefault{\guillemetleft}{%
2050   \UseTextSymbol{OT1}{\guillemetleft}}
2051 \ProvideTextCommandDefault{\guillemetright}{%
2052   \UseTextSymbol{OT1}{\guillemetright}}
2053 \ProvideTextCommandDefault{\guillemotleft}{%
2054   \UseTextSymbol{OT1}{\guillemotleft}}
2055 \ProvideTextCommandDefault{\guillemotright}{%
2056   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 2057 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2058   \ifmmode
2059     <%
2060   \else
2061     \save@sf@q{\nobreak
2062       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2063   \fi}
2064 \ProvideTextCommand{\guilsinglright}{OT1}{%
2065   \ifmmode
2066     >%
2067   \else
2068     \save@sf@q{\nobreak
2069       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2070   \fi}
```

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

```
2071 \ProvideTextCommandDefault{\guilsinglleft}{%
```

```

2072 \UseTextSymbol{OT1}{\guilsinglleft}}
2073 \ProvideTextCommandDefault{\guilsinglright}{%
2074 \UseTextSymbol{OT1}{\guilsinglright}}

```

#### 4.12.2 Letters

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

```

2075 \DeclareTextCommand{\ij}{OT1}{%
2076 i\kern-0.02em\bbl@allowhyphens j}
2077 \DeclareTextCommand{\IJ}{OT1}{%
2078 I\kern-0.02em\bbl@allowhyphens J}
2079 \DeclareTextCommand{\ij}{T1}{\char188}
2080 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2081 \ProvideTextCommandDefault{\ij}{%
2082 \UseTextSymbol{OT1}{\ij}}
2083 \ProvideTextCommandDefault{\IJ}{%
2084 \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

2085 \def\crrtic@{\hrule height0.1ex width0.3em}
2086 \def\crttic@{\hrule height0.1ex width0.33em}
2087 \def\ddj@{%
2088 \setbox0\hbox{d}\dimen@=\ht0
2089 \advance\dimen@lex
2090 \dimen@.45\dimen@
2091 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2092 \advance\dimen@ii.5ex
2093 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2094 \def\DDJ@{%
2095 \setbox0\hbox{D}\dimen@=.55\ht0
2096 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2097 \advance\dimen@ii.15ex % correction for the dash position
2098 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2099 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2100 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2101 %
2102 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2103 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2104 \ProvideTextCommandDefault{\dj}{%
2105 \UseTextSymbol{OT1}{\dj}}
2106 \ProvideTextCommandDefault{\DJ}{%
2107 \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.

```

2108 \DeclareTextCommand{\SS}{OT1}{SS}
2109 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

#### 4.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with \ProvideTextCommandDefault, but this is very likely not required because their definitions are based on encoding-dependent macros.

`\glq` The ‘german’ single quotes.

```
\grq 2110 \ProvideTextCommandDefault{\glq}{%
      2111 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2112 \ProvideTextCommand{\grq}{T1}{%
2113 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2114 \ProvideTextCommand{\grq}{TU}{%
2115 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2116 \ProvideTextCommand{\grq}{OT1}{%
2117 \save@sf@q{\kern-.0125em
2118 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2119 \kern.07em\relax}}
2120 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2121 \ProvideTextCommandDefault{\glqq}{%
      2122 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2123 \ProvideTextCommand{\grqq}{T1}{%
2124 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2125 \ProvideTextCommand{\grqq}{TU}{%
2126 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2127 \ProvideTextCommand{\grqq}{OT1}{%
2128 \save@sf@q{\kern-.07em
2129 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2130 \kern.07em\relax}}
2131 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2132 \ProvideTextCommandDefault{\flq}{%
      2133 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2134 \ProvideTextCommandDefault{\frq}{%
      2135 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2136 \ProvideTextCommandDefault{\flqq}{%
      2137 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2138 \ProvideTextCommandDefault{\frqq}{%
      2139 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

#### 4.12.4 Umlauts and tremas

The command `\` needs to have a different effect for different languages. For German for instance, the ‘umlaut’ should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

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

```
2140 \def\umlauthigh{%
2141 \def\bbl@umlauta##1{\leavevmode\bgroup%
2142 \accent\csname\fontencoding dqpos\endcsname
2143 ##1\bbl@allowhyphens\egroup}%
2144 \let\bbl@umlaute\bbl@umlauta}
2145 \def\umlautlow{%
2146 \def\bbl@umlauta{\protect\lower@umlaut}}
2147 \def\umlautelower{%
2148 \def\bbl@umlaute{\protect\lower@umlaut}}
2149 \umlauthigh
```

`\lower@umlaut` The command `\lower@umlaut` is used to position the `\` closer to the letter. We want the umlaut character lowered, nearer to the letter. To do this we need an extra *(dimen)* register.

```
2150 \expandafter\ifx\csname U@D\endcsname\relax
2151   \csname newdimen\endcsname\U@D
2152 \fi
```

The following code fools  $\TeX$ 's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

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

```
2153 \def\lower@umlaut#1{%
2154   \leavevmode\bgroup
2155     \U@D 1ex%
2156     {\setbox\z@\hbox{%
2157       \char\csname\fontencoding dqpos\endcsname}%
2158       \dimen@ -.45ex\advance\dimen@\ht\z@
2159       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2160     \accent\csname\fontencoding dqpos\endcsname
2161     \fontdimen5\font\U@D #1%
2162   \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).

```
2163 \AtBeginDocument{%
2164   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2165   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2166   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2167   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2168   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2169   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2170   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2171   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2172   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2173   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2174   \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`.

```
2175 \ifx\l@english\undefined
2176   \chardef\l@english\z@
2177 \fi
2178 % The following is used to cancel rules in ini files (see Amharic).
2179 \ifx\l@unhyphenated\undefined
2180   \newlanguage\l@unhyphenated
2181 \fi
```

### 4.13 Layout

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

```
2182 \bbl@trace{Bidi layout}
2183 \providecommand\IfBabelLayout[3]{#3}%
2184 {-core}
2185 \newcommand\BabelPatchSection[1]{%
2186   \@ifundefined{#1}{}{%
```

```

2187 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2188 \@namedef{#1}{%
2189 \ifstar{\bbl@presec@#1}%
2190 {\@dblarg{\bbl@presec@x{#1}}}}%
2191 \def\bbl@presec@x#1[#2]#3{%
2192 \bbl@exp{%
2193 \\\select@language@x{\bbl@main@language}%
2194 \\\bbl@cs{sspre@#1}%
2195 \\\bbl@cs{ss@#1}%
2196 [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2197 {\foreignlanguage{\language}{\unexpanded{#3}}}%
2198 \\\select@language@x{\language}}%
2199 \def\bbl@presec@s#1#2{%
2200 \bbl@exp{%
2201 \\\select@language@x{\bbl@main@language}%
2202 \\\bbl@cs{sspre@#1}%
2203 \\\bbl@cs{ss@#1}*%
2204 {\foreignlanguage{\language}{\unexpanded{#2}}}%
2205 \\\select@language@x{\language}}%
2206 \IfBabelLayout{sectioning}%
2207 {\BabelPatchSection{part}%
2208 \BabelPatchSection{chapter}%
2209 \BabelPatchSection{section}%
2210 \BabelPatchSection{subsection}%
2211 \BabelPatchSection{subsubsection}%
2212 \BabelPatchSection{paragraph}%
2213 \BabelPatchSection{subparagraph}%
2214 \def\babel@toc#1{%
2215 \select@language@x{\bbl@main@language}}}%
2216 \IfBabelLayout{captions}%
2217 {\BabelPatchSection{caption}}}%
2218 \<+core>

```

#### 4.14 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.

```

2219 \bbl@trace{Input engine specific macros}
2220 \ifcase\bbl@engine
2221 \input txtbabel.def
2222 \or
2223 \input luababel.def
2224 \or
2225 \input xebabel.def
2226 \fi
2227 \providecommand\babelfont{\bbl@error{only-lua-xe}}{}{}{}
2228 \providecommand\babelprehyphenation{\bbl@error{only-lua}}{}{}{}
2229 \ifx\babelposthyphenation\undefined
2230 \let\babelposthyphenation\babelprehyphenation
2231 \let\babelpatterns\babelprehyphenation
2232 \let\babelcharproperty\babelprehyphenation
2233 \fi

```

#### 4.15 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.

```

2234 \</package | core>
2235 \<*package>
2236 \bbl@trace{Creating languages and reading ini files}

```

```

2237 \let\bbl@extend@ini@gobble
2238 \newcommand\babelprovide[2][]{%
2239   \let\bbl@savelangname\language
2240   \edef\bbl@savelocaleid{\the\localeid}%
2241   % Set name and locale id
2242   \edef\language{#2}%
2243   \bbl@id@assign
2244   % Initialize keys
2245   \bbl@vforeach{captions,date,import,main,script,language,%
2246     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2247     mapdigits,intraspaces,intrapenalty,onchar,transforms,alpha,%
2248     Alph,labels,labels*,calendar,date,casing,interchar}%
2249     {\bbl@csarg\let{KVP@##1}\@nnil}%
2250   \global\let\bbl@release@transforms\@empty
2251   \global\let\bbl@release@casing\@empty
2252   \let\bbl@calendars\@empty
2253   \global\let\bbl@inidata\@empty
2254   \global\let\bbl@extend@ini@gobble
2255   \global\let\bbl@included@inis\@empty
2256   \gdef\bbl@key@list{;}%
2257   \bbl@forkv{#1}{%
2258     \in@{/}{##1}% With /, (re)sets a value in the ini
2259     \ifin@
2260       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2261       \bbl@renewinikey##1\@{##2}%
2262     \else
2263       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2264         \bbl@error{unknown-provide-key}{##1}{}%
2265       \fi
2266       \bbl@csarg\def{KVP@##1}{##2}%
2267     \fi}%
2268   \chardef\bbl@howloaded=0:none;1:ldf without ini;2:ini
2269   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2270   % == init ==
2271   \ifx\bbl@screset\@undefined
2272     \bbl@ldfinit
2273   \fi
2274   % == date (as option) ==
2275   % \ifx\bbl@KVP@date\@nnil\else
2276   % \fi
2277   % ==
2278   \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2279   \ifcase\bbl@howloaded
2280     \let\bbl@lbkflag\@empty % new
2281   \else
2282     \ifx\bbl@KVP@hyphenrules\@nnil\else
2283       \let\bbl@lbkflag\@empty
2284     \fi
2285     \ifx\bbl@KVP@import\@nnil\else
2286       \let\bbl@lbkflag\@empty
2287     \fi
2288   \fi
2289   % == import, captions ==
2290   \ifx\bbl@KVP@import\@nnil\else
2291     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2292     {\ifx\bbl@initload\relax
2293       \begingroup
2294         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2295         \bbl@input@texini{#2}%
2296       \endgroup
2297     \else
2298       \xdef\bbl@KVP@import{\bbl@initload}%
2299     \fi}%

```

```

2300     {}%
2301     \let\bbl@KVP@date\@empty
2302 \fi
2303 \let\bbl@KVP@captions@\bbl@KVP@captions % TODO. A dirty hack
2304 \ifx\bbl@KVP@captions\@nnil
2305     \let\bbl@KVP@captions\bbl@KVP@import
2306 \fi
2307 % ==
2308 \ifx\bbl@KVP@transforms\@nnil\else
2309     \bbl@replace\bbl@KVP@transforms{ }{,}%
2310 \fi
2311 % == Load ini ==
2312 \ifcase\bbl@howloaded
2313     \bbl@provide@new{#2}%
2314 \else
2315     \bbl@ifblank{#1}%
2316         {}% With \bbl@load@basic below
2317         {\bbl@provide@renew{#2}}%
2318 \fi
2319 % == include == TODO
2320 % \ifx\bbl@included@inis\@empty\else
2321 %     \bbl@replace\bbl@included@inis{ }{,}%
2322 %     \bbl@foreach\bbl@included@inis{%
2323 %         \openin\bbl@readstream=babel-##1.ini
2324 %         \bbl@extend@ini{#2}%
2325 %         \closein\bbl@readstream
2326 %     \fi
2327 % Post tasks
2328 % -----
2329 % == subsequent calls after the first provide for a locale ==
2330 \ifx\bbl@inidata\@empty\else
2331     \bbl@extend@ini{#2}%
2332 \fi
2333 % == ensure captions ==
2334 \ifx\bbl@KVP@captions\@nnil\else
2335     \bbl@ifunset{\bbl@extracaps@#2}%
2336         {\bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}}%
2337         {\bbl@exp{\bbl@babelensure[exclude=\\today,
2338             include=\[bbl@extracaps@#2]]{#2}}}%
2339     \bbl@ifunset{\bbl@ensure@\language}%
2340         {\bbl@exp{%
2341             \\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2342                 \\foreignlanguage{\language}%
2343                 {###1}}}%
2344         }{}%
2345     \bbl@exp{%
2346         \\bbl@tglobal\<bbl@ensure@\language>%
2347         \\bbl@tglobal\<bbl@ensure@\language\space>}%
2348 \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.

```

2349 \bbl@load@basic{#2}%
2350 % == script, language ==
2351 % Override the values from ini or defines them
2352 \ifx\bbl@KVP@script\@nnil\else
2353     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2354 \fi
2355 \ifx\bbl@KVP@language\@nnil\else
2356     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2357 \fi
2358 \ifcase\bbl@engine\or

```



```

2359 \bbl@ifunset{bbl@chrng@{language}}{%
2360   {\directlua{
2361     Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2362 \fi
2363 % == onchar ==
2364 \ifx\bbl@KVP@onchar\@nnil\else
2365   \bbl@luahyphenate
2366   \bbl@exp{%
2367     \\\AddToHook{env/document/before}{\select@language{#2}}}%
2368   \directlua{
2369     if Babel.locale_mapped == nil then
2370       Babel.locale_mapped = true
2371       Babel.linebreaking.add_before(Babel.locale_map, 1)
2372       Babel.loc_to_scr = {}
2373       Babel.chr_to_loc = Babel.chr_to_loc or {}
2374     end
2375     Babel.locale_props[\the\localeid].letters = false
2376   }%
2377   \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2378   \ifin@
2379     \directlua{
2380       Babel.locale_props[\the\localeid].letters = true
2381     }%
2382   \fi
2383   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2384   \ifin@
2385     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2386       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2387     \fi
2388     \bbl@exp{\bbl@add\bbl@starthyphens
2389       {\bbl@patterns@lua{language}}}%
2390     % TODO - error/warning if no script
2391     \directlua{
2392       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2393         Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcpr}']
2394         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@{language}}\space
2395       end
2396     }%
2397   \fi
2398   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2399   \ifin@
2400     \bbl@ifunset{bbl@lsys@{language}}{\bbl@provide@lsys{language}}}%
2401     \bbl@ifunset{bbl@wdir@{language}}{\bbl@provide@dirs{language}}}%
2402     \directlua{
2403       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2404         Babel.loc_to_scr[\the\localeid] =
2405           Babel.script_blocks['\bbl@cl{sbcpr}']
2406       end}%
2407   \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
2408     \AtBeginDocument{%
2409       \bbl@patchfont{\bbl@mapselect}%
2410       {\selectfont}}%
2411     \def\bbl@mapselect{%
2412       \let\bbl@mapselect\relax
2413       \edef\bbl@prefontid{\fontid\font}}%
2414     \def\bbl@mapdir##1{%
2415       \begingroup
2416         \setbox\z@\hbox{% Force text mode
2417           \def\language{##1}%
2418           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2419           \bbl@switchfont
2420           \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2421             \directlua{

```

```

2422             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2423             ['/\/bbl@prefontid'] = \fontid\font\space}%
2424         \fi}%
2425     \endgroup}%
2426 \fi
2427     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language name}}}%
2428 \fi
2429 % TODO - catch non-valid values
2430 \fi
2431 % == mapfont ==
2432 % For bidi texts, to switch the font based on direction
2433 \ifx\bbl@KVP@mapfont\@nnil\else
2434     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
2435     {\bbl@error{unknown-mapfont}}}%
2436     \bbl@ifunset{\bbl@sys@language name}{\bbl@provide@sys{\language name}}}%
2437     \bbl@ifunset{\bbl@wdir@language name}{\bbl@provide@dirs{\language name}}}%
2438     \ifx\bbl@mapselect\undefined % TODO. See onchar.
2439         \AtBeginDocument{%
2440             \bbl@patchfont{\bbl@mapselect}}%
2441             {\selectfont}}%
2442         \def\bbl@mapselect{%
2443             \let\bbl@mapselect\relax
2444             \edef\bbl@prefontid{\fontid\font}}%
2445         \def\bbl@mapdir##1{%
2446             {\def\language name{##1}%
2447             \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2448             \bbl@switchfont
2449             \directlua{Babel.fontmap
2450             [\the\csname bbl@wdir@##1\endcsname]%
2451             [\bbl@prefontid]=\fontid\font}}}%
2452     \fi
2453     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language name}}}%
2454 \fi
2455 % == Line breaking: intraspace, intrapenalty ==
2456 % For CJK, East Asian, Southeast Asian, if interspace in ini
2457 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2458     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2459 \fi
2460 \bbl@provide@intraspace
2461 % == Line breaking: CJK quotes == TODO -> @extras
2462 \ifcase\bbl@engine\or
2463     \bbl@xin@{/c}{/\/bbl@cl{\lnbrk}}}%
2464     \ifin@
2465         \bbl@ifunset{\bbl@quote@language name}}}%
2466         {\directlua{
2467             Babel.locale_props[\the\localeid].cjk_quotes = {}
2468             local cs = 'op'
2469             for c in string.utfvalues(
2470                 [[\csname bbl@quote@language name\endcsname]]) do
2471                 if Babel.cjk_characters[c].c == 'qu' then
2472                     Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2473                 end
2474                 cs = ( cs == 'op') and 'cl' or 'op'
2475             end
2476         }}%
2477     \fi
2478 \fi
2479 % == Line breaking: justification ==
2480 \ifx\bbl@KVP@justification\@nnil\else
2481     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2482 \fi
2483 \ifx\bbl@KVP@linebreaking\@nnil\else
2484     \bbl@xin@{\bbl@KVP@linebreaking,}%

```

```

2485     {,elongated,kashida,cjk,padding,unhyphenated,}%
2486 \ifin@
2487 \bbl@csarg\xdef
2488     {lnbrk@\language\name}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2489 \fi
2490 \fi
2491 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2492 \ifin@else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2493 \ifin@\bbl@arabicjust\fi
2494 \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2495 \ifin@\AtBeginDocument{\@nameuse{\bbl@tibetanjust}}\fi
2496 % == Line breaking: hyphenate.other.(locale|script) ==
2497 \ifx\bbl@lbcflag\empty
2498     \bbl@ifunset{\bbl@hyotl@\language\name}{}%
2499     {\bbl@csarg\bbl@replace{\hyotl@\language\name}{ }{,}%
2500     \bbl@startcommands*{\language\name}{}%
2501     \bbl@csarg\bbl@foreach{\hyotl@\language\name}{%
2502     \ifcase\bbl@engine
2503     \ifnum##1<257
2504     \SetHyphenMap{\BabelLower{##1}{##1}}%
2505     \fi
2506     \else
2507     \SetHyphenMap{\BabelLower{##1}{##1}}%
2508     \fi}%
2509     \bbl@endcommands}%
2510 \bbl@ifunset{\bbl@hyots@\language\name}{}%
2511 {\bbl@csarg\bbl@replace{\hyots@\language\name}{ }{,}%
2512 \bbl@csarg\bbl@foreach{\hyots@\language\name}{%
2513 \ifcase\bbl@engine
2514 \ifnum##1<257
2515 \global\lccode##1=##1\relax
2516 \fi
2517 \else
2518 \global\lccode##1=##1\relax
2519 \fi}}%
2520 \fi
2521 % == Counters: maparabic ==
2522 % Native digits, if provided in ini (TeX level, xe and lua)
2523 \ifcase\bbl@engine\else
2524     \bbl@ifunset{\bbl@dgnat@\language\name}{}%
2525     {\expandafter\ifx\csname\bbl@dgnat@\language\name\endcsname\empty\else
2526     \expandafter\expandafter\expandafter
2527     \bbl@setdigits\csname\bbl@dgnat@\language\name\endcsname
2528     \ifx\bbl@KVP@maparabic\@nnil\else
2529     \ifx\bbl@latinarabic\undefined
2530     \expandafter\let\expandafter\@arabic
2531     \csname\bbl@counter@\language\name\endcsname
2532     \else % ie, if layout=counters, which redefines \@arabic
2533     \expandafter\let\expandafter\bbl@latinarabic
2534     \csname\bbl@counter@\language\name\endcsname
2535     \fi
2536     \fi
2537 \fi}%
2538 \fi
2539 % == Counters: mapdigits ==
2540 % > luababel.def
2541 % == Counters: alph, Alph ==
2542 \ifx\bbl@KVP@alph\@nnil\else
2543     \bbl@exp{%
2544     \\ \bbl@add\<\bbl@preextras@\language\name>{%
2545     \\ \babel@save\\ \@alph
2546     \let\\ \@alph\<\bbl@cntr@\bbl@KVP@alph @\language\name>}}%
2547 \fi

```

```

2548 \ifx\bbbl@KVP@Alph@\nnil\else
2549 \bbbl@exp{%
2550 \\\bbbl@add\<\bbbl@preextras@\language\>{%
2551 \\\babel@save\\\@Alph
2552 \let\\\@Alph\<\bbbl@cntr@\bbbl@KVP@Alph @\language\>}}%
2553 \fi
2554 % == Casing ==
2555 \bbbl@release@casing
2556 \ifx\bbbl@KVP@casing@\nnil\else
2557 \bbbl@csarg\xdef{casing@\language}%
2558 {\@nameuse{\bbbl@casing@\language}\bbbl@maybextx\bbbl@KVP@casing}%
2559 \fi
2560 % == Calendars ==
2561 \ifx\bbbl@KVP@calendar@\nnil
2562 \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}%
2563 \fi
2564 \def\bbbl@tempe##1##2\@{%% Get first calendar
2565 \def\bbbl@tempa{##1}}%
2566 \bbbl@exp{\bbbl@tempe\bbbl@KVP@calendar\space\\\@}%
2567 \def\bbbl@tempe##1.##2.##3\@{%%
2568 \def\bbbl@tempc{##1}%
2569 \def\bbbl@tempb{##2}}%
2570 \expandafter\bbbl@tempe\bbbl@tempa..\@
2571 \bbbl@csarg\edef{calpr@\language}%
2572 \ifx\bbbl@tempc@empty\else
2573 calendar=\bbbl@tempc
2574 \fi
2575 \ifx\bbbl@tempb@empty\else
2576 ,variant=\bbbl@tempb
2577 \fi}%
2578 % == engine specific extensions ==
2579 % Defined in XXXbabel.def
2580 \bbbl@provide@extra{#2}%
2581 % == require.babel in ini ==
2582 % To load or reload the babel-*.tex, if require.babel in ini
2583 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2584 \bbbl@ifunset{\bbbl@rqtex@\language}%
2585 {\expandafter\ifx\csname \bbbl@rqtex@\language\endcsname\@empty\else
2586 \let\BabelBeforeIni\@gobbletwo
2587 \chardef\atcatcode=\catcode\@
2588 \catcode\@=11\relax
2589 \def\CurrentOption{#2}%
2590 \bbbl@input@texini{\bbbl@cs{rqtex@\language}}%
2591 \catcode\@=\atcatcode
2592 \let\atcatcode\relax
2593 \global\bbbl@csarg\let{rqtex@\language}\relax
2594 \fi}%
2595 \bbbl@foreach\bbbl@calendars{%
2596 \bbbl@ifunset{\bbbl@ca##1}{%
2597 \chardef\atcatcode=\catcode\@
2598 \catcode\@=11\relax
2599 \InputIfFileExists{babel-ca-##1.tex}{}}%
2600 \catcode\@=\atcatcode
2601 \let\atcatcode\relax}%
2602 {}}%
2603 \fi
2604 % == frenchspacing ==
2605 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2606 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2607 \ifin@
2608 \bbbl@extras@wrap{\bbbl@pre@fs}%
2609 {\bbbl@pre@fs}%
2610 {\bbbl@post@fs}%

```

```

2611 \fi
2612 % == transforms ==
2613 % > luababel.def
2614 \def\CurrentOption{#2}%
2615 \@nameuse{bbl@icsave@#2}%
2616 % == main ==
2617 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2618   \let\language\@nameuse{bbl@savelangname}
2619   \chardef\localeid\@nameuse{bbl@savelocaleid}\relax
2620 \fi
2621 % == hyphenrules (apply if current) ==
2622 \ifx\bbl@KVP@hyphenrules\@nnil\else
2623   \ifnum\bbl@savelocaleid=\localeid
2624     \language\@nameuse{l@\language}%
2625   \fi
2626 \fi}

```

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

```

2627 \def\bbl@provide@new#1{%
2628   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2629   \@namedef{extras#1}{}%
2630   \@namedef{noextras#1}{}%
2631   \bbl@startcommands*{#1}{captions}%
2632   \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2633     \def\bbl@tempb##1{% elt for \bbl@captionslist
2634       \ifx##1\@nnil\else
2635         \bbl@exp{%
2636           \\SetString\\##1{%
2637             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2638           \expandafter\bbl@tempb
2639         \fi}%
2640     \expandafter\bbl@tempb\bbl@captionslist\@nnil
2641   \else
2642     \ifx\bbl@initoload\relax
2643       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2644     \else
2645       \bbl@read@ini{\bbl@initoload}2% % Same
2646     \fi
2647   \fi
2648   \StartBabelCommands*{#1}{date}%
2649   \ifx\bbl@KVP@date\@nnil
2650     \bbl@exp{%
2651       \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
2652   \else
2653     \bbl@savetoday
2654     \bbl@savedate
2655   \fi
2656   \bbl@endcommands
2657   \bbl@load@basic{#1}%
2658   % == hyphenmins == (only if new)
2659   \bbl@exp{%
2660     \gdef\<#1hyphenmins>{%
2661       {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2662       {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}%
2663   % == hyphenrules (also in renew) ==
2664   \bbl@provide@hyphens{#1}%
2665   \ifx\bbl@KVP@main\@nnil\else
2666     \expandafter\main@language\expandafter{#1}%
2667   \fi}
2668 %
2669 \def\bbl@provide@renew#1{%
2670   \ifx\bbl@KVP@captions\@nnil\else

```

```

2671 \StartBabelCommands*{#1}{captions}%
2672 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2673 \EndBabelCommands
2674 \fi
2675 \ifx\bbl@KVP@date\@nnil\else
2676 \StartBabelCommands*{#1}{date}%
2677 \bbl@savetoday
2678 \bbl@savedate
2679 \EndBabelCommands
2680 \fi
2681 % == hyphenrules (also in new) ==
2682 \ifx\bbl@lbfkflag\@empty
2683 \bbl@provide@hyphens{#1}%
2684 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)

```

2685 \def\bbl@load@basic#1{%
2686 \ifcase\bbl@howloaded\or\or
2687 \ifcase\csname bbl@llevel\language\endcsname
2688 \bbl@csarg\let\lname@\language\relax
2689 \fi
2690 \fi
2691 \bbl@ifunset{\bbl@lname@#1}%
2692 {\def\BabelBeforeIni##1##2{%
2693 \begingroup
2694 \let\bbl@ini@captions@aux\@gobbletwo
2695 \def\bbl@inidate ###1.###2.###3.###4\relax ###5###6}%
2696 \bbl@read@ini{##1}1%
2697 \ifx\bbl@initoload\relax\endinput\fi
2698 \endgroup}%
2699 \begingroup % boxed, to avoid extra spaces:
2700 \ifx\bbl@initoload\relax
2701 \bbl@input@texini{##1}%
2702 \else
2703 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2704 \fi
2705 \endgroup}%
2706 {}%

```

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.

```

2707 \def\bbl@provide@hyphens#1{%
2708 \@tempcnta\m@ne % a flag
2709 \ifx\bbl@KVP@hyphenrules\@nnil\else
2710 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2711 \bbl@foreach\bbl@KVP@hyphenrules{%
2712 \ifnum\@tempcnta=\m@ne % if not yet found
2713 \bbl@ifsamestring{##1}{+}%
2714 {\bbl@carg\addlanguage{l@##1}}%
2715 }%
2716 \bbl@ifunset{l@##1}% After a possible +
2717 {}%
2718 {\@tempcnta\@nameuse{l@##1}}%
2719 \fi}%
2720 \ifnum\@tempcnta=\m@ne
2721 \bbl@warning{%
2722 Requested 'hyphenrules' for '\language' not found:\\%
2723 \bbl@KVP@hyphenrules.\\%
2724 Using the default value. Reported}%
2725 \fi
2726 \fi
2727 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found

```

```

2728 \ifx\bbbl@KVP@captions@@\@nnil % TODO. Hackish. See above.
2729 \bbbl@ifunset{\bbbl@hyphr@#1}{}% use value in ini, if exists
2730 {\bbbl@exp{\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
2731 {}%
2732 {\bbbl@ifunset{\l@bbbl@c{hyphr}}}%
2733 {}% if hyphenrules found:
2734 {\@tempcnta\@nameuse{\l@bbbl@c{hyphr}}}%
2735 \fi
2736 \fi
2737 \bbbl@ifunset{\l@#1}%
2738 {\ifnum\@tempcnta=\m@ne
2739 \bbbl@carg\adddialect{\l@#1}\language
2740 \else
2741 \bbbl@carg\adddialect{\l@#1}\@tempcnta
2742 \fi}%
2743 {\ifnum\@tempcnta=\m@ne\else
2744 \global\bbbl@carg\chardef{\l@#1}\@tempcnta
2745 \fi}}

```

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

```

2746 \def\bbbl@input@texini#1{%
2747 \bbbl@bsphack
2748 \bbbl@exp{%
2749 \catcode`\%%=14 \catcode`\%%=0
2750 \catcode`\%{=1 \catcode`\%{=2
2751 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2752 \catcode`\%%=\the\catcode`\%\relax
2753 \catcode`\%{=\the\catcode`\%{relax
2754 \catcode`\%{=\the\catcode`\%\relax
2755 \catcode`\%{=\the\catcode`\%\relax}%
2756 \bbbl@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 \bbbl@read@ini.

```

2757 \def\bbbl@inline#1\bbbl@inline{%
2758 \ifnextchar[\bbbl@iniset{\ifnextchar\bbbl@iniskip\bbbl@inistore}#1\@@}% ]
2759 \def\bbbl@iniset[#1]#2\@@{\def\bbbl@section{#1}}
2760 \def\bbbl@iniskip#1\@@{% if starts with ;
2761 \def\bbbl@inistore#1=#2\@@{% full (default)
2762 \bbbl@trim@def\bbbl@tempa{#1}%
2763 \bbbl@trim\toks@{#2}%
2764 \bbbl@xin@{\bbbl@section/\bbbl@tempa;}{\bbbl@key@list}%
2765 \ifin@else
2766 \bbbl@xin@{,identification/include.}%
2767 {\bbbl@section/\bbbl@tempa}%
2768 \ifin@\xdef\bbbl@included@inis{\the\toks@}\fi
2769 \bbbl@exp{%
2770 \\\g@addto@macro\bbbl@inidata{%
2771 \\\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
2772 \fi}
2773 \def\bbbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbbl@read@ini)
2774 \bbbl@trim@def\bbbl@tempa{#1}%
2775 \bbbl@trim\toks@{#2}%
2776 \bbbl@xin@{.identification.}{\bbbl@section.}%
2777 \ifin@
2778 \bbbl@exp{\\\g@addto@macro\bbbl@inidata{%
2779 \\\bbbl@elt{identification}{\bbbl@tempa}{\the\toks@}}}%
2780 \fi}

```

Now, the 'main loop', which **must be executed inside a group**. At this point, \bbbl@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.

```

2781 \def\bbl@loop@ini{%
2782   \loop
2783     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2784     \endlinechar\m@ne
2785     \read\bbl@readstream to \bbl@line
2786     \endlinechar\^^M
2787     \ifx\bbl@line\@empty\else
2788       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2789     \fi
2790   \repeat}
2791 \ifx\bbl@readstream\@undefined
2792   \csname newread\endcsname\bbl@readstream
2793 \fi
2794 \def\bbl@read@ini#1#2{%
2795   \global\let\bbl@extend@ini\@gobble
2796   \openin\bbl@readstream=babel-#1.ini
2797   \ifeof\bbl@readstream
2798     \bbl@error{no-ini-file}{#1}{}{}%
2799   \else
2800     % == Store ini data in \bbl@inidata ==
2801     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2802     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2803     \bbl@info{Importing
2804               \ifcase#2font and identification \or basic \fi
2805               data for \language\name}%
2806     from babel-#1.ini. Reported}%
2807     \ifnum#2=\z@
2808       \global\let\bbl@inidata\@empty
2809       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2810     \fi
2811     \def\bbl@section{identification}%
2812     \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2813     \bbl@inistore load.level=#2\\@@
2814     \bbl@loop@ini
2815     % == Process stored data ==
2816     \bbl@csarg\xdef{lini@\language}{#1}%
2817     \bbl@read@ini@aux
2818     % == 'Export' data ==
2819     \bbl@ini@exports{#2}%
2820     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2821     \global\let\bbl@inidata\@empty
2822     \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
2823     \bbl@tglobal\bbl@ini@loaded
2824   \fi
2825   \closein\bbl@readstream}
2826 \def\bbl@read@ini@aux{%
2827   \let\bbl@savestrings\@empty
2828   \let\bbl@savetoday\@empty
2829   \let\bbl@savestate\@empty
2830   \def\bbl@elt##1##2##3{%
2831     \def\bbl@section{##1}%
2832     \in@{=date.}{=##1}% Find a better place
2833     \ifin@
2834       \bbl@ifunset{bbl@inikv@##1}%
2835       {\bbl@ini@calendar{##1}}%
2836     }%
2837   \fi
2838   \bbl@ifunset{bbl@inikv@##1}{}%
2839   {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2840   \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.

```

2841 \def\bbl@extend@ini@aux#1{%
2842   \bbl@startcommands*{#1}{captions}%
2843   % Activate captions/... and modify exports
2844   \bbl@csarg\def\inikv@captions.licr}##1##2{%
2845     \setlocalecaption{#1}{##1}{##2}}%
2846   \def\bbl@inikv@captions##1##2{%
2847     \bbl@ini@captions@aux{##1}{##2}}%
2848   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2849   \def\bbl@exportkey##1##2##3{%
2850     \bbl@ifunset{\bbl@kv@##2}{}%
2851     {\expandafter\ifx\csname\bbl@kv@##2\endcsname\@empty\else
2852      \bbl@exp{\global\let<\bbl@##1@language>\<\bbl@kv@##2>}}%
2853     \fi}}%
2854   % As with \bbl@read@ini, but with some changes
2855   \bbl@read@ini@aux
2856   \bbl@ini@exports\tw@
2857   % Update inidata@lang by pretending the ini is read.
2858   \def\bbl@elt##1##2##3{%
2859     \def\bbl@section{##1}%
2860     \bbl@iniline##2=##3\bbl@iniline}%
2861     \csname\bbl@inidata@#1\endcsname
2862     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2863   \StartBabelCommands*{#1}{date}% And from the import stuff
2864   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2865   \bbl@savetoday
2866   \bbl@savestate
2867   \bbl@endcommands}

```

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

```

2868 \def\bbl@ini@calendar#1{%
2869   \lowercase{\def\bbl@tempa{=1=}}%
2870   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2871   \bbl@replace\bbl@tempa{=date.}{}%
2872   \in@{.licr}{#1=}%
2873   \ifin@
2874     \ifcase\bbl@engine
2875       \bbl@replace\bbl@tempa{.licr}{}%
2876     \else
2877       \let\bbl@tempa\relax
2878     \fi
2879   \fi
2880   \ifx\bbl@tempa\relax\else
2881     \bbl@replace\bbl@tempa{=}{}%
2882     \ifx\bbl@tempa\@empty\else
2883       \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2884     \fi
2885     \bbl@exp{%
2886       \def<\bbl@inikv@#1>####1####2{%
2887         \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2888     \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).

```

2889 \def\bbl@renewinikey#1/#2\@#3{%
2890   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2891   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2892   \bbl@trim\toks@{#3}% value
2893   \bbl@exp{%
2894     \edef\\bbl@key@list{\bbl@key@list\bbl@tempa/\bbl@tempb;}%

```

```

2895    \\g@addto@macro\\bbl@inidata{%
2896    \\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.

```

2897 \def\bbl@exportkey#1#2#3{%
2898   \bbl@ifunset{\bbl@kv@#2}%
2899   {\bbl@csarg\gdef{#1@{language}\language}{#3}}%
2900   {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2901     \bbl@csarg\gdef{#1@{language}\language}{#3}%
2902     \else
2903     \bbl@exp{\global\let<bbl@#1@{language}>\<bbl@kv@#2>}%
2904     \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.

```

2905 \def\bbl@iniwarning#1{%
2906   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2907   {\bbl@warning{%
2908     From babel-\bbl@cs{lini@{language}\language}.ini:\\%
2909     \bbl@cs{kv@identification.warning#1}\\%
2910     Reported }}%
2911 %
2912 \let\bbl@release@transforms\@empty
2913 \let\bbl@release@casing\@empty
2914 \def\bbl@ini@exports#1{%
2915   % Identification always exported
2916   \bbl@iniwarning{}%
2917   \ifcase\bbl@engine
2918     \bbl@iniwarning{.pdflatex}%
2919   \or
2920     \bbl@iniwarning{.lualatex}%
2921   \or
2922     \bbl@iniwarning{.xelatex}%
2923   \fi%
2924   \bbl@exportkey{llevel}{identification.load.level}{}%
2925   \bbl@exportkey{elname}{identification.name.english}{}%
2926   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2927     {\csname bbl@elname@{language}\endcsname}}%
2928   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2929   % Somewhat hackish. TODO:
2930   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2931   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2932   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2933   \bbl@exportkey{esname}{identification.script.name}{}%
2934   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2935     {\csname bbl@esname@{language}\endcsname}}%
2936   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2937   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2938   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2939   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2940   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2941   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2942   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2943   % Also maps bcp47 -> language
2944   \ifbbl@bcptoname
2945     \bbl@csarg\xdef{bcp@map@{bbl@cl{tbc}}}{\language}%
2946   \fi
2947   \ifcase\bbl@engine\or
2948     \directlua{%

```

```

2949      Babel.locale_props[\the\bbl@cs{id@@\languagename}].script
2950      = '\bbl@cl{sbcpr}'%
2951 \fi
2952 % Conditional
2953 \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
2954   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2955   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2956   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2957   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2958   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2959   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2960   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2961   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2962   \bbl@exportkey{intsp}{typography.intraspaces}{}%
2963   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2964   \bbl@exportkey{chrng}{characters.ranges}{}%
2965   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2966   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2967   \ifnum#1=\tw@          % only (re)new
2968     \bbl@exportkey{rqtex}{identification.require.babel}{}%
2969     \bbl@toglobal\bbl@savetoday
2970     \bbl@toglobal\bbl@savestate
2971     \bbl@savestrings
2972 \fi
2973 \fi}

```

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

```

2974 \def\bbl@inikv#1#2{%      key=value
2975   \toks@{#2}%             This hides #'s from ini values
2976   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2977 \let\bbl@inikv@identification\bbl@inikv
2978 \let\bbl@inikv@date\bbl@inikv
2979 \let\bbl@inikv@typography\bbl@inikv
2980 \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.

```

2981 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2982 \def\bbl@inikv@characters#1#2{%
2983   \bbl@ifsamestring{#1}{casing}% eg, casing = uV
2984   {\bbl@exp{%
2985     \\\g@addto@macro\\\bbl@release@casing{%
2986       \\\bbl@casemapping}{\languagename}{\unexpanded{#2}}}%
2987     {\in@{casing.}{#1}% eg, casing.Uv = uV
2988     \ifin@
2989       \lowercase{\def\bbl@tempb{#1}}%
2990       \bbl@replace\bbl@tempb{casing.}{}%
2991       \bbl@exp{\\g@addto@macro\\\bbl@release@casing{%
2992         \\\bbl@casemapping
2993         {\\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}%
2994       \else
2995       \bbl@inikv{#1}{#2}%
2996     \fi}}

```

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

```

2997 \def\bbl@inikv@counters#1#2{%
2998   \bbl@ifsamestring{#1}{digits}%
2999   {\bbl@error{digits-is-reserved}{}}}%
3000   {}%

```

```

3001 \def\bbl@tempc{#1}%
3002 \bbl@trim@def{\bbl@tempb*}{#2}%
3003 \in@{.l$}{#1$}%
3004 \ifin@
3005   \bbl@replace\bbl@tempc{.l}{}%
3006   \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3007     \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3008 \fi
3009 \in@{.F.}{#1}%
3010 \ifin@ \else \in@{.S.}{#1} \fi
3011 \ifin@
3012   \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3013 \else
3014   \toks@{}% Required by \bbl@builddifcase, which returns \bbl@tempa
3015   \expandafter\bbl@builddifcase\bbl@tempb* \ \ % Space after \
3016   \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3017 \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.

```

3018 \ifcase\bbl@engine
3019   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3020     \bbl@ini@captions@aux{#1}{#2}}
3021 \else
3022   \def\bbl@inikv@captions#1#2{%
3023     \bbl@ini@captions@aux{#1}{#2}}
3024 \fi

```

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

```

3025 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3026   \bbl@replace\bbl@tempa{.template}{}%
3027   \def\bbl@toreplace{#1}{}%
3028   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3029   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3030   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3031   \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}%
3032   \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3033   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3034   \ifin@
3035     \@nameuse{\bbl@patch\bbl@tempa}%
3036     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3037   \fi
3038   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3039   \ifin@
3040     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3041     \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3042       \\bbl@ifunset{\bbl@\bbl@tempa fmt@\\languagename}%
3043       {[fnum@\bbl@tempa]}%
3044       {\\@nameuse{\bbl@\bbl@tempa fmt@\\languagename}}}%
3045     \fi}
3046 \def\bbl@ini@captions@aux#1#2{%
3047   \bbl@trim@def\bbl@tempa{#1}%
3048   \bbl@xin@{.template}{\bbl@tempa}%
3049   \ifin@
3050     \bbl@ini@captions@template{#2}\languagename
3051   \else
3052     \bbl@ifblank{#2}%
3053     {\bbl@exp{%
3054       \toks@{\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3055     {\bbl@trim\toks@{#2}}%
3056     \bbl@exp{%
3057       \\bbl@add\\bbl@savestrings{%
3058         \\SetString\<\bbl@tempa name>{\the\toks@}}}%

```

```

3059 \toks@expandafter{\bbl@captionslist}%
3060 \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3061 \ifin@ \else
3062 \bbl@exp{%
3063 \\\bbl@add\<bbl@extracaps@language>{\<\bbl@tempa name>}%
3064 \\\bbl@tglobal\<bbl@extracaps@language>}%
3065 \fi
3066 \fi}

```

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

```

3067 \def\bbl@list@the{%
3068 part,chapter,section,subsection,subsubsection,paragraph,%
3069 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3070 table,page,footnote,mpfootnote,mpfn}
3071 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3072 \bbl@ifunset{\bbl@map@#1@language}%
3073 {\@nameuse{#1}}%
3074 {\@nameuse{\bbl@map@#1@language}}}%
3075 \def\bbl@inikv@labels#1#2{%
3076 \in@{.map}{#1}%
3077 \ifin@
3078 \ifx\bbl@KVP@labels\@nnil\else
3079 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3080 \ifin@
3081 \def\bbl@tempc{#1}%
3082 \bbl@replace\bbl@tempc{.map}{}%
3083 \in@{,#2,},{arabic,roman,Roman,alph,Alph,fnsymbol,}%
3084 \bbl@exp{%
3085 \gdef\<bbl@map@\bbl@tempc @language>%
3086 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3087 \bbl@foreach\bbl@list@the{%
3088 \bbl@ifunset{the##1}{}%
3089 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3090 \bbl@exp{%
3091 \\\bbl@sreplace\<the##1>%
3092 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3093 \\\bbl@sreplace\<the##1>%
3094 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3095 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3096 \toks@expandafter\expandafter\expandafter{%
3097 \csname the##1\endcsname}%
3098 \expandafter\edef\csname the##1\endcsname{\the\toks@}}%
3099 \fi}}%
3100 \fi
3101 \fi
3102 %
3103 \else
3104 %
3105 % The following code is still under study. You can test it and make
3106 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3107 % language dependent.
3108 \in@{enumerate.}{#1}%
3109 \ifin@
3110 \def\bbl@tempa{#1}%
3111 \bbl@replace\bbl@tempa{enumerate.}{}%
3112 \def\bbl@toreplace{#2}%
3113 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3114 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3115 \bbl@replace\bbl@toreplace{[]}{\endcsname}}%
3116 \toks@expandafter{\bbl@toreplace}%
3117 % TODO. Execute only once:
3118 \bbl@exp{%
3119 \\\bbl@add\<extras\language>%

```

```

3120      \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3121      \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3122      \\bbl@tglobal\<extras\language>%
3123      \fi
3124      \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.

```

3125 \def\bbl@chapttype{chapter}
3126 \ifx\@makechapterhead\@undefined
3127   \let\bbl@patchchapter\relax
3128 \else\ifx\thechapter\@undefined
3129   \let\bbl@patchchapter\relax
3130 \else\ifx\ps@headings\@undefined
3131   \let\bbl@patchchapter\relax
3132 \else
3133   \def\bbl@patchchapter{%
3134     \global\let\bbl@patchchapter\relax
3135     \gdef\bbl@chfmt{%
3136       \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3137       {\@chapapp\space\thechapter}
3138       {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}
3139     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3140     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3141     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3142     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3143     \bbl@tglobal\appendix
3144     \bbl@tglobal\ps@headings
3145     \bbl@tglobal\chaptermark
3146     \bbl@tglobal\@makechapterhead}
3147     \let\bbl@patchappendix\bbl@patchchapter
3148   \fi\fi\fi
3149 \ifx\@part\@undefined
3150   \let\bbl@patchpart\relax
3151 \else
3152   \def\bbl@patchpart{%
3153     \global\let\bbl@patchpart\relax
3154     \gdef\bbl@partformat{%
3155       \bbl@ifunset{\bbl@partfmt@\language}%
3156       {\partname\nobreakspace\thepart}
3157       {\@nameuse{\bbl@partfmt@\language}}}
3158     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3159     \bbl@tglobal\@part}
3160   \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

```

3161 \let\bbl@calendar\@empty
3162 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3163 \def\bbl@localedate#1#2#3#4{%
3164   \begingroup
3165     \edef\bbl@they{#2}%
3166     \edef\bbl@them{#3}%
3167     \edef\bbl@thed{#4}%
3168     \edef\bbl@tempe{%
3169       \bbl@ifunset{\bbl@calpr@\language}{\bbl@cl{calpr}},%
3170       #1}%
3171     \bbl@replace\bbl@tempe{ }{}%
3172     \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3173     \bbl@replace\bbl@tempe{convert}{convert=}%
3174     \let\bbl@ld@calendar\@empty
3175     \let\bbl@ld@variant\@empty

```

```

3176 \let\bbl@ld@convert\relax
3177 \def\bbl@tempb##1=##2\@@{\@namedef\bbl@ld@##1}{##2}}%
3178 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3179 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3180 \ifx\bbl@ld@calendar\@empty\else
3181 \ifx\bbl@ld@convert\relax\else
3182 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3183 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3184 \fi
3185 \fi
3186 \@nameuse\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3187 \edef\bbl@calendar{% Used in \month..., too
3188 \bbl@ld@calendar
3189 \ifx\bbl@ld@variant\@empty\else
3190 .\bbl@ld@variant
3191 \fi}%
3192 \bbl@cased
3193 {\@nameuse\bbl@date@\language @\bbl@calendar}%
3194 \bbl@they\bbl@them\bbl@thed}%
3195 \endgroup}
3196 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3197 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3198 \bbl@trim@def\bbl@tempa{#1.#2}%
3199 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3200 {\bbl@trim@def\bbl@tempa{#3}%
3201 \bbl@trim\toks@{#5}%
3202 \@temptokena\expandafter{\bbl@savestate}%
3203 \bbl@exp{% Reverse order - in ini last wins
3204 \def\\bbl@savestate{%
3205 \\SetString<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3206 \the\@temptokena}}}%
3207 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3208 {\lowercase{\def\bbl@tempb{#6}}}%
3209 \bbl@trim@def\bbl@toreplace{#5}%
3210 \bbl@TG@@date
3211 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3212 \ifx\bbl@savetoday\@empty
3213 \bbl@exp{% TODO. Move to a better place.
3214 \\AfterBabelCommands{%
3215 \def<\language date>{\protect<\language date >}%
3216 \\newcommand<\language date >[4][]{%
3217 \\bbl@usedategroupttrue
3218 \<bbl@ensure@\language >{%
3219 \\localedate[####1]{####2}{####3}{####4}}}%
3220 \def\\bbl@savetoday{%
3221 \\SetString\\today{%
3222 \<\language date>[convert]%
3223 {\the\year}{\the\month}{\the\day}}}%
3224 \fi}%
3225 {}}}}

```

**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).

```

3226 \let\bbl@calendar\@empty
3227 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3228 \@nameuse\bbl@ca@#2}#1\@@}
3229 \newcommand\babelDateSpace{\nobreakspace}
3230 \newcommand\babelDateDot{\. \@ % TODO. \let instead of repeating
3231 \newcommand\babelDated[1]{\number#1}}
3232 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}}

```

```

3233 \newcommand\BabelDateM[1]{\number#1}
3234 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3235 \newcommand\BabelDateMMM[1]{%
3236   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3237 \newcommand\BabelDateY[1]{\number#1}%
3238 \newcommand\BabelDateYY[1]{%
3239   \ifnum#1<10 0\number#1 %
3240   \else\ifnum#1<100 \number#1 %
3241   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3242   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3243   \else
3244     \bbl@error{limit-two-digits}{\number#1}%
3245   \fi\fi\fi\fi}
3246 \newcommand\BabelDateYYYY[1]{\number#1} % TODO - add leading 0
3247 \newcommand\BabelDateU[1]{\number#1}%
3248 \def\bbl@replace@finish@iii#1{%
3249   \bbl@exp{\def\#1####1####2####3\the\toks@}}
3250 \def\bbl@TG@date{%
3251   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3252   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3253   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3254   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3255   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3256   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3257   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3258   \bbl@replace\bbl@toreplace{[y]}{\BabelDateY{####1}}%
3259   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateYY{####1}}%
3260   \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateYYY{####1}}%
3261   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}%
3262   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecntr[####1]}%
3263   \bbl@replace\bbl@toreplace{[U]}{\bbl@datecntr[####1]}%
3264   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecntr[####2]}%
3265   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecntr[####3]}%
3266   \bbl@replace@finish@iii\bbl@toreplace}
3267 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3268 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

```

### Transforms.

```

3269 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3270 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3271 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3272   #1[#2]{#3}{#4}{#5}}
3273 \begingroup % A hack. TODO. Don't require an specific order
3274   \catcode`\%=12
3275   \catcode`\&=14
3276   \gdef\bbl@transforms#1#2#3{%&
3277     \directlua{
3278       local str = {[#2]}
3279       str = str:gsub('%.%d+%.%d+$', '')
3280       token.set_macro('babeltempa', str)
3281     }&
3282     \def\babeltempc{}&
3283     \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&
3284     \ifin@ \else
3285       \bbl@xin@{:,\babeltempa,},{,\bbl@KVP@transforms,}&
3286     \fi
3287     \ifin@
3288       \bbl@foreach\bbl@KVP@transforms{%&
3289         \bbl@xin@{:,\babeltempa,},{,##1,}&
3290         \ifin@ & font:font:transform syntax
3291         \directlua{
3292           local t = {}
3293           for m in string.gmatch('##1'..'':', '(-.):') do

```



```

3294         table.insert(t, m)
3295     end
3296     table.remove(t)
3297     token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3298 }&%
3299 \fi}&%
3300 \in@{.0$}{#2$}&%
3301 \ifin@
3302     \directlua{&% (\attribute) syntax
3303         local str = string.match([[ \bbl@KVP@transforms]],
3304             '%([^(%-)]-)[^%)]-\babeltempa')
3305         if str == nil then
3306             token.set_macro('babeltempb', '')
3307         else
3308             token.set_macro('babeltempb', ', attribute=' .. str)
3309         end
3310     }&%
3311     \toks@{#3}&%
3312     \bbl@exp{&%
3313         \\g@addto@macro\\bbl@release@transforms{&%
3314             \relax &% Closes previous \bbl@transforms@aux
3315             \\bbl@transforms@aux
3316             \\\#1{label=\babeltempa\babeltempb\babeltempc}&%
3317             {\language\the\toks@}}&%
3318     \else
3319         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3320     \fi
3321 \fi}
3322 \endgroup

```

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

```

3323 \def\bbl@provide@lsys#1{%
3324     \bbl@ifunset{bbl@lname@#1}%
3325     {\bbl@load@info{#1}}%
3326     {%
3327         \bbl@csarg\let{lsys@#1}\@empty
3328         \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3329         \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3330         \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3331         \bbl@ifunset{bbl@lname@#1}{%
3332             {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3333         \ifcase\bbl@engine\or\or
3334             \bbl@ifunset{bbl@prehc@#1}{%
3335                 {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3336                 {%
3337                     {\ifx\bbl@xenoxyph\undefined
3338                         \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3339                     \ifx\AtBeginDocument\notprerr
3340                         \expandafter\@secondoftwo % to execute right now
3341                     \fi
3342                     \AtBeginDocument{%
3343                         \bbl@patchfont{\bbl@xenoxyph}%
3344                         {\expandafter\select@language\expandafter{\language\the\toks@}}}%
3345                     \fi}}%
3346             \fi
3347             \bbl@csarg\bbl@to@global{lsys@#1}}
3348 \def\bbl@xenoxyph@d{%
3349     \bbl@ifset{bbl@prehc@\language\the\toks@}%
3350     {\ifnum\hyphenchar\font=\defaultshyphenchar
3351         \iffontchar\font\bbl@cl{prehc}\relax
3352         \hyphenchar\font\bbl@cl{prehc}\relax
3353     \else\iffontchar\font"200B

```

```

3354         \hyphenchar\font"200B
3355     \else
3356         \bbl@warning
3357             {Neither 0 nor ZERO WIDTH SPACE are available\\%
3358               in the current font, and therefore the hyphen\\%
3359               will be printed. Try changing the fontspec's\\%
3360               'HyphenChar' to another value, but be aware\\%
3361               this setting is not safe (see the manual).\\%
3362               Reported}%
3363         \hyphenchar\font\defaultshyphenchar
3364     \fi\fi
3365 \fi}%
3366 {\hyphenchar\font\defaultshyphenchar}}
3367 % \fi}

```

```

3368 \def\bbl@load@info#1{%
3369   \def\BabelBeforeIni##1##2{%
3370     \begingroup
3371       \bbl@read@ini{##1}0%
3372       \endinput           % babel- .tex may contain onlypreamble's
3373       \endgroup}%         boxed, to avoid extra spaces:
3374   {\bbl@input@texini{#1}}}
```

[illegible]

```
3406 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
```

```

3407 \ifx\#1% % \ before, in case #1 is multiletter
3408 \bbl@exp{%
3409 \def\#1\bbl@tempa###1{%
3410 \<ifcase>###1\space\the\toks@<else>\@ctrerr<fi>}%
3411 \else
3412 \toks@<expandafter{\the\toks@<or #1>%
3413 \expandafter\bbl@builddifcase
3414 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3415 \newcommand\localecnumeral[2]{\bbl@cs{cntnr@#1\language}\{#2}}
3416 \def\bbl@localecnumeral#1#2{\localecnumeral{#2}{#1}}
3417 \newcommand\localecounter[2]{%
3418 \expandafter\bbl@localecnumeral
3419 \expandafter{\number\csname c@#2\endcsname}{#1}}
3420 \def\bbl@alphnumeral#1#2{%
3421 \expandafter\bbl@alphnumeral@i\language#2 76543210\@{#1}}
3422 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@{#9}%
3423 \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
3424 \bbl@alphnumeral@ii{#9}000000#1\or
3425 \bbl@alphnumeral@ii{#9}00000#1#2\or
3426 \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3427 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3428 \bbl@alphnum@invalid{>9999}%
3429 \fi}
3430 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3431 \bbl@ifunset{bbl@cntnr@#1.F.\number#5#6#7#8\language}%
3432 {\bbl@cs{cntnr@#1.4\language}\{#5}}
3433 {\bbl@cs{cntnr@#1.3\language}\{#6}}
3434 {\bbl@cs{cntnr@#1.2\language}\{#7}}
3435 {\bbl@cs{cntnr@#1.1\language}\{#8}}
3436 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3437 \bbl@ifunset{bbl@cntnr@#1.S.321\language}\{#9}}
3438 {\bbl@cs{cntnr@#1.S.321\language}\{#9}}
3439 \fi}%
3440 {\bbl@cs{cntnr@#1.F.\number#5#6#7#8\language}\{#9}}
3441 \def\bbl@alphnum@invalid#1{%
3442 \bbl@error{alphabetic-too-large}{#1}}

```

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

```

3443 \def\bbl@localeinfo#1#2{%
3444 \bbl@ifunset{bbl@info@#2}{#1}%
3445 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}\{#1}}
3446 {\bbl@cs{csname bbl@info@#2\endcsname @\language}}
3447 \newcommand\localeinfo[1]{%
3448 \ifx*#1\empty % TODO. A bit hackish to make it expandable.
3449 \bbl@afterelse\bbl@localeinfo}%
3450 \else
3451 \bbl@localeinfo
3452 {\bbl@error{no-ini-info}}}%
3453 {#1}%
3454 \fi}
3455 % \namedef{bbl@info@name.locale}\{lname}
3456 \namedef{bbl@info@tag.ini}\{lini}
3457 \namedef{bbl@info@name.english}\{elname}
3458 \namedef{bbl@info@name.opentype}\{lname}
3459 \namedef{bbl@info@tag.bcp47}\{tbcp}
3460 \namedef{bbl@info@language.tag.bcp47}\{lbcp}
3461 \namedef{bbl@info@tag.opentype}\{lotf}

```

```

3462 \@namedef{bbl@info@script.name}{esname}
3463 \@namedef{bbl@info@script.name.opentype}{sname}
3464 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3465 \@namedef{bbl@info@script.tag.opentype}{sotf}
3466 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3467 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3468 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3469 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3470 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}

```

$\LaTeX$  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.

```

3471 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3472   \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3473 \else
3474   \def\bbl@utftocode#1{\expandafter`\string#1}
3475 \fi
3476 % Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
3477 % expandable (|\bbl@ifsamestring| isn't).
3478 \providecommand\BCPdata{}
3479 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3480   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3481   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3482     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3483     {\bbl@bcpdata@ii#6}\bbl@main@language}%
3484     {\bbl@bcpdata@ii#1#2#3#4#5#6}\language}%
3485   \def\bbl@bcpdata@ii#1#2{%
3486     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3487     {\bbl@error{unknown-ini-field}{#1}{}}%
3488     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3489     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3490 \fi
3491 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3492 \newcommand\BabelUppercaseMapping[3]{%
3493   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3494 \newcommand\BabelTitlecaseMapping[3]{%
3495   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3496 \newcommand\BabelLowercaseMapping[3]{%
3497   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

```

The parser for casing and casing.(variant).

```

3498 \def\bbl@casemapping#1#2#3{% 1:variant
3499   \def\bbl@tempa##1 ##2{% Loop
3500     \bbl@casemapping@i{##1}%
3501     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3502   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3503   \def\bbl@tempe{0}% Mode (upper/lower...)
3504   \def\bbl@tempc{#3}% Casing list
3505   \expandafter\bbl@tempa\bbl@tempc\@empty}
3506 \def\bbl@casemapping@i#1{%
3507   \def\bbl@tempb{#1}%
3508   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3509     \@nameuse{regex_replace_all:nnN}%
3510     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\0}}\bbl@tempb
3511   \else
3512     \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb % TODO. needed?
3513   \fi
3514   \expandafter\bbl@casemapping@ii\bbl@tempb\@{}
3515 \def\bbl@casemapping@ii#1#2#3\@{%
3516   \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3517   \ifin@
3518     \edef\bbl@tempe{%
3519       \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%

```

```

3520 \else
3521   \ifcase\bb@tempe\relax
3522     \DeclareUppercaseMapping[\bb@templ]{\bb@uftocode{#1}}{#2}%
3523     \DeclareLowercaseMapping[\bb@templ]{\bb@uftocode{#2}}{#1}%
3524   \or
3525     \DeclareUppercaseMapping[\bb@templ]{\bb@uftocode{#1}}{#2}%
3526   \or
3527     \DeclareLowercaseMapping[\bb@templ]{\bb@uftocode{#1}}{#2}%
3528   \or
3529     \DeclareTitlecaseMapping[\bb@templ]{\bb@uftocode{#1}}{#2}%
3530   \fi
3531 \fi}

```

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

```

3532 <<(*More package options)>> ≡
3533 \DeclareOption{ensureinfo=off}{}
3534 <</More package options>>
3535 \let\bb@ensureinfo@gobble
3536 \newcommand\BabelEnsureInfo{%
3537   \ifx\InputIfFileExists\undefined\else
3538     \def\bb@ensureinfo##1{%
3539       \bb@ifunset{bb@lname@##1}{\bb@load@info{##1}}{}}%
3540   \fi
3541   \bb@foreach\bb@loaded{%
3542     \let\bb@ensuring\@empty % Flag used in a couple of babel-*.tex files
3543     \def\languagename{##1}%
3544     \bb@ensureinfo{##1}}}%
3545 \ifpackagewith{babel}{ensureinfo=off}{}%
3546 {\AtEndOfPackage{% Test for plain.
3547   \ifx\undefined\bb@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3548 \newcommand\getlocaleproperty{%
3549   \ifstar\bb@getproperty@s\bb@getproperty@x}
3550 \def\bb@getproperty@s#1#2#3{%
3551   \let#1\relax
3552   \def\bb@elt##1##2##3{%
3553     \bb@ifsamestring{##1/##2}{#3}%
3554     {\providecommand#1{##3}%
3555     \def\bb@elt####1####2####3{}}}%
3556   {}}%
3557   \bb@cs{inidata@#2}}%
3558 \def\bb@getproperty@x#1#2#3{%
3559   \bb@getproperty@s{#1}{#2}{#3}%
3560   \ifx#1\relax
3561     \bb@error{unknown-locale-key}{#1}{#2}{#3}%
3562   \fi}
3563 \let\bb@ini@loaded\@empty
3564 \newcommand\LocaleForEach{\bb@foreach\bb@ini@loaded}
3565 \def\ShowLocaleProperties#1{%
3566   \typeout{}%
3567   \typeout{*** Properties for language '#1' ***}
3568   \def\bb@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3569   \@nameuse{bb@inidata@#1}%
3570   \typeout{*****}}

```

## 5 Adjusting the Babel behavior

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

```

3571 \newcommand\babeladjust[1]{% TODO. Error handling.

```

```

3572 \bbl@forkv{#1}{%
3573   \bbl@ifunset{bbl@ADJ@##1@##2}%
3574   {\bbl@cs{ADJ@##1}{##2}}%
3575   {\bbl@cs{ADJ@##1@##2}}}%
3576 %
3577 \def\bbl@adjust@lua#1#2{%
3578   \ifvmode
3579     \ifnum\currentgrouplevel=\z@
3580       \directlua{ Babel.#2 }%
3581       \expandafter\expandafter\expandafter\@gobble
3582     \fi
3583   \fi
3584   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3585 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3586   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3587 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3588   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3589 \@namedef{bbl@ADJ@bidi.text@on}{%
3590   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3591 \@namedef{bbl@ADJ@bidi.text@off}{%
3592   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3593 \@namedef{bbl@ADJ@bidi.math@on}{%
3594   \let\bbl@noamsmath\@empty}
3595 \@namedef{bbl@ADJ@bidi.math@off}{%
3596   \let\bbl@noamsmath\relax}
3597 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3598   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3599 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3600   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3601 %
3602 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3603   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3604 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3605   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3606 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3607   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3608 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3609   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3610 \@namedef{bbl@ADJ@justify.arabic@on}{%
3611   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3612 \@namedef{bbl@ADJ@justify.arabic@off}{%
3613   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3614 %
3615 \def\bbl@adjust@layout#1{%
3616   \ifvmode
3617     #1%
3618     \expandafter\@gobble
3619   \fi
3620   {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3621 \@namedef{bbl@ADJ@layout.tabular@on}{%
3622   \ifnum\bbl@tabular@mode=\tw@
3623     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3624   \else
3625     \chardef\bbl@tabular@mode\@ne
3626   \fi}
3627 \@namedef{bbl@ADJ@layout.tabular@off}{%
3628   \ifnum\bbl@tabular@mode=\tw@
3629     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3630   \else
3631     \chardef\bbl@tabular@mode\z@
3632   \fi}
3633 \@namedef{bbl@ADJ@layout.lists@on}{%
3634   \bbl@adjust@layout{\let\list\bbl@NL@list}}

```

```

3635 \@namedef{bbl@ADJ@layout.lists@off}{%
3636   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3637 %
3638 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3639   \bbl@bcpallowedtrue}
3640 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3641   \bbl@bcpallowedfalse}
3642 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3643   \def\bbl@bcp@prefix{#1}}
3644 \def\bbl@bcp@prefix{bcp47-}
3645 \@namedef{bbl@ADJ@autoload.options}#1{%
3646   \def\bbl@autoload@options{#1}}
3647 \let\bbl@autoload@bcptoptions\@empty
3648 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3649   \def\bbl@autoload@bcptoptions{#1}}
3650 \newif\ifbbl@bcptoname
3651 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3652   \bbl@bcptonametrue}
3653 \BabelEnsureInfo{
3654 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3655   \bbl@bcptonamefalse}
3656 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3657   \directlua{ Babel.ignore_pre_char = function(node)
3658     return (node.lang == \the\csname l@nohyphenation\endcsname)
3659   end }}
3660 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3661   \directlua{ Babel.ignore_pre_char = function(node)
3662     return false
3663   end }}
3664 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3665   \def\bbl@ignoreinterchar{%
3666     \ifnum\language=\l@nohyphenation
3667       \expandafter\@gobble
3668     \else
3669       \expandafter\@firstofone
3670     \fi}}
3671 \@namedef{bbl@ADJ@interchar.disable@off}{%
3672   \let\bbl@ignoreinterchar\@firstofone}
3673 \@namedef{bbl@ADJ@select.write@shift}{%
3674   \let\bbl@restorelastskip\relax
3675   \def\bbl@savelastskip{%
3676     \let\bbl@restorelastskip\relax
3677     \ifvmode
3678       \ifdim\lastskip=\z@
3679         \let\bbl@restorelastskip\nobreak
3680       \else
3681         \bbl@exp{%
3682           \def\\bbl@restorelastskip{%
3683             \skip@=\the\lastskip
3684             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3685         \fi
3686       \fi}}
3687 \@namedef{bbl@ADJ@select.write@keep}{%
3688   \let\bbl@restorelastskip\relax
3689   \let\bbl@savelastskip\relax}
3690 \@namedef{bbl@ADJ@select.write@omit}{%
3691   \AddBabelHook{babel-select}{beforestart}{%
3692     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3693 \let\bbl@restorelastskip\relax
3694 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3695 \@namedef{bbl@ADJ@select.encoding@off}{%
3696   \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.

```
3697 <<{*More package options}>> ≡
3698 \DeclareOption{safe=none}{\let\bbl@opt@safe\empty}
3699 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3700 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3701 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3702 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3703 <</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.

```
3704 \bbl@trace{Cross referencing macros}
3705 \ifx\bbl@opt@safe\empty\else % ie, if 'ref' and/or 'bib'
3706   \def\@newl@bel#1#2#3{%
3707     \@safe@activestrue
3708     \bbl@ifunset{#1@#2}%
3709       \relax
3710     {\gdef\@multiplelabels{%
3711       \@latex@warning@no@line{There were multiply-defined labels}}%
3712     \@latex@warning@no@line{Label `#2' multiply defined}}%
3713     \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.

```
3714 \CheckCommand*\@testdef[3]{%
3715   \def\reserved@a{#3}%
3716   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3717   \else
3718     \@tempswatrue
3719   \fi}
```

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

```
3720 \def\@testdef#1#2#3{% TODO. With @samestring?
3721   \@safe@activestrue
3722   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3723   \def\bbl@tempb{#3}%
3724   \@safe@activesfalse
3725   \ifx\bbl@tempa\relax
3726   \else
3727     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3728   \fi
3729   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3730   \ifx\bbl@tempa\bbl@tempb
3731   \else
3732     \@tempswatrue
3733   \fi}
3734 \fi
```



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

```

3735 \bbl@xin@{R}\bbl@opt@safe
3736 \ifin@
3737 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3738 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3739 {\expandafter\strip@prefix\meaning\ref}%
3740 \ifin@
3741 \bbl@redefine\@kernel@ref#1{%
3742   \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetruefalse}
3743 \bbl@redefine\@kernel@pageref#1{%
3744   \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetruefalse}
3745 \bbl@redefine\@kernel@sref#1{%
3746   \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetruefalse}
3747 \bbl@redefine\@kernel@spageref#1{%
3748   \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetruefalse}
3749 \else
3750 \bbl@redefineroquest\ref#1{%
3751   \@safe@activetrue\org@ref{#1}\@safe@activetruefalse}
3752 \bbl@redefineroquest\pageref#1{%
3753   \@safe@activetrue\org@pageref{#1}\@safe@activetruefalse}
3754 \fi
3755 \else
3756 \let\org@ref\ref
3757 \let\org@pageref\pageref
3758 \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.

```

3759 \bbl@xin@{B}\bbl@opt@safe
3760 \ifin@
3761 \bbl@redefine\@citex[#1]#2{%
3762   \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetruefalse
3763   \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.

```

3764 \AtBeginDocument{%
3765   \@ifpackageloaded{natbib}{%

```

Notice that we use `\def` here instead of `\bbl@redefine` because `\org@@citex` is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).

(Recent versions of `natbib` change dynamically `\@citex`, so PR4087 doesn't seem fixable in a simple way. Just load `natbib` before.)

```

3766   \def\@citex[#1][#2]#3{%
3767     \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activetruefalse
3768     \org@@citex[#1][#2]{\bbl@tempa}}%
3769   }{}

```

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

```

3770 \AtBeginDocument{%
3771   \@ifpackageloaded{cite}{%
3772     \def\@citex[#1]#2{%
3773       \@safe@activetrue\org@@citex[#1][#2]\@safe@activetruefalse}%
3774     }{}

```

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

```
3775 \bbl@redefine\nocite#1{%
3776   \@safe@activestruer\org@nocite{#1}\@safe@activesfalse}
```

`\bibcite` The macro that is used in the .aux file to define citation labels. When packages such as `natbib` or `cite` are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where `\@safe@activestruer` 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.

```
3777 \bbl@redefine\bibcite{%
3778   \bbl@cite@choice
3779   \bibcite}
```

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

```
3780 \def\bbl@bibcite#1#2{%
3781   \org@bibcite{#1}\@safe@activesfalse#2}}
```

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

```
3782 \def\bbl@cite@choice{%
3783   \global\let\bibcite\bbl@bibcite
3784   \ifpackageloaded{natbib}\global\let\bibcite\org@bibcite}%
3785   \ifpackageloaded{cite}\global\let\bibcite\org@bibcite}%
3786   \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.

```
3787 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3788 \bbl@redefine\@bibitem#1{%
3789   \@safe@activestruer\org@bibitem{#1}\@safe@activesfalse}
3790 \else
3791   \let\org@nocite\nocite
3792   \let\org@citex\citex
3793   \let\org@bibcite\bibcite
3794   \let\org@bibitem\@bibitem
3795 \fi
```

## 5.2 Marks

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

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

```
3796 \bbl@trace{Marks}
3797 \IfBabelLayout{sectioning}
3798   {\ifx\bbl@opt@headfoot\@nnil
3799     \g@addto@macro\@resetactivechars{%
3800       \set@typeset@protect
3801       \expandafter\select@language\x\expandafter{\bbl@main@language}%
3802       \let\protect\noexpand
3803       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3804         \edef\thepage{%
3805           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3806       \fi}%
```

```

3807 \fi}
3808 {\ifbbl@single\else
3809 \bbl@ifunset{markright }\bbl@redefine\bbl@redefineroobust
3810 \markright#1{%
3811 \bbl@ifblank{#1}%
3812 {\org@markright{}}}%
3813 {\toks@{#1}%
3814 \bbl@exp{%
3815 \\\org@markright{\\protect\\foreignlanguage{\language}\language}%
3816 {\\\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

3817 \ifx\mkboth\markboth
3818 \def\bbl@tempc{\let\mkboth\markboth}%
3819 \else
3820 \def\bbl@tempc{%
3821 \fi
3822 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefineroobust
3823 \markboth#1#2{%
3824 \protected@edef\bbl@tempb##1{%
3825 \protect\foreignlanguage
3826 {\language}\protect\bbl@restore@actives##1}}%
3827 \bbl@ifblank{#1}%
3828 {\toks@{}}%
3829 {\toks@\expandafter{\bbl@tempb{#1}}}%
3830 \bbl@ifblank{#2}%
3831 {\@temptokena{}}%
3832 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3833 \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3834 \bbl@tempc
3835 \fi} % end ifbbl@single, end \IfBabelLayout

```

## 5.3 Preventing clashes with other packages

### 5.3.1 `ifthen`

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

```

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

```

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

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

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

```

3836 \bbl@trace{Preventing clashes with other packages}
3837 \ifx\org@ref\undefined\else
3838 \bbl@xin@{R}\bbl@opt@safe
3839 \ifin@
3840 \AtBeginDocument{%
3841 \@ifpackageloaded{ifthen}{%

```

```

3842      \bbl@redefine@long\ifthenelse#1#2#3{%
3843      \let\bbl@temp@pref\pageref
3844      \let\pageref\org@pageref
3845      \let\bbl@temp@ref\ref
3846      \let\ref\org@ref
3847      \@safe@activetrue
3848      \org@ifthenelse{#1}%
3849      {\let\pageref\bbl@temp@pref
3850      \let\ref\bbl@temp@ref
3851      \@safe@activesfalse
3852      #2}%
3853      {\let\pageref\bbl@temp@pref
3854      \let\ref\bbl@temp@ref
3855      \@safe@activesfalse
3856      #3}%
3857      }%
3858      }{}%
3859      }
3860 \fi

```

### 5.3.2 varioref

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

```

3861 \AtBeginDocument{%
3862   \ifpackageloaded{varioref}{%
3863     \bbl@redefine\@vpageref#1[#2]#3{%
3864       \@safe@activetrue
3865       \org@@@vpageref{#1}[#2]#3}%
3866       \@safe@activesfalse}%
3867     \bbl@redefine\vrefpagemum#1#2{%
3868       \@safe@activetrue
3869       \org@vrefpagemum{#1}#2}%
3870       \@safe@activesfalse}%

```

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

```

3871   \expandafter\def\csname Ref \endcsname#1{%
3872     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3873   }{}%
3874   }
3875 \fi

```

### 5.3.3 hhline

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

```

3876 \AtEndOfPackage{%
3877   \AtBeginDocument{%
3878     \ifpackageloaded{hhline}%
3879     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3880       \else
3881         \makeatletter
3882         \def\@currname{hhline}\input{hhline.sty}\makeatother
3883       \fi}%
3884     {}}}}

```

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

```

3885 \def\substitutefontfamily#1#2#3{%
3886   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3887   \immediate\write15{%
3888     \string\ProvidesFile{#1#2.fd}%
3889     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3890     \space generated font description file]^J
3891     \string\DeclareFontFamily{#1}{#2}{^^J
3892     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3893     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3894     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3895     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3896     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3897     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3898     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3899     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3900   }%
3901   \closeout15
3902 }
3903 \@onlypreamble\substitutefontfamily

```

## 5.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\TeX$  and  $\LaTeX$  always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings are currently stored in `\fontenc@load@list`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

```

\ensureascii
3904 \bbl@trace{Encoding and fonts}
3905 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3906 \newcommand\BabelNonText{TS1,T3,TS3}
3907 \let\org@TeX\TeX
3908 \let\org@LaTeX\LaTeX
3909 \let\ensureascii\@firstofone
3910 \let\asciientcoding\@empty
3911 \AtBeginDocument{%
3912   \def\@elt#1{,#1,}%
3913   \edef\bbl@tempa{\expandafter\@gobbletwo\fontenc@load@list}%
3914   \let\@elt\relax
3915   \let\bbl@tempb\@empty
3916   \def\bbl@tempc{OT1}%
3917   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3918     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3919   \bbl@foreach\bbl@tempa{%
3920     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3921     \ifin@
3922       \def\bbl@tempb{#1}% Store last non-ascii
3923     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3924     \ifin@
3925       \def\bbl@tempc{#1}% Store last ascii
3926     \fi
3927   \fi}%
3928   \ifx\bbl@tempb\@empty\else
3929     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3930     \ifin@
3931       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3932     \fi
3933     \let\asciientcoding\bbl@tempc
3934     \renewcommand\ensureascii[1]{%

```

```

3935     {\fontencoding{\asciientencoding}\selectfont#1}}%
3936     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3937     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3938     \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.

```

3939 \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.

```

3940 \AtBeginDocument{%
3941   \@ifpackageloaded{fontspec}%
3942   {\xdef\latinencoding{%
3943     \ifx\UTFencname\undefined
3944       EU\ifcase\bbl@engine\or2\or1\fi
3945     \else
3946       \UTFencname
3947     \fi}}%
3948   {\gdef\latinencoding{OT1}%
3949     \ifx\cf@encoding\bbl@t@one
3950       \xdef\latinencoding{\bbl@t@one}%
3951     \else
3952       \def\@elt#1{,#1,}%
3953       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontencload@list}%
3954       \let\@elt\relax
3955       \bbl@xin@{,T1,}\bbl@tempa
3956       \ifin@
3957         \xdef\latinencoding{\bbl@t@one}%
3958       \fi
3959     \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.

```

3960 \DeclareRobustCommand{\latintext}{%
3961   \fontencoding{\latinencoding}\selectfont
3962   \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.

```

3963 \ifx\@undefined\DeclareTextFontCommand
3964   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3965 \else
3966   \DeclareTextFontCommand{\textlatin}{\latintext}
3967 \fi

```

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

```

3968 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

## 5.5 Basic bidi support

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

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This `babel` module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been

copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at ARABI (by Youssef Jabri), which is compatible with babel.

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

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

```

3969 \bbl@trace{Loading basic (internal) bidi support}
3970 \ifodd\bbl@engine
3971 \else % TODO. Move to txtbabel
3972   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3973     \bbl@error{bidi-only-lua}{\}\}\}%
3974     \let\bbl@beforeforeign\leavevmode
3975     \AtEndOfPackage{%
3976       \EnableBabelHook{babel-bidi}%
3977       \bbl@xebidipar}
3978   \fi\fi
3979   \def\bbl@loadxebidi#1{%
3980     \ifx\RTLfootnotetext\@undefined
3981       \AtEndOfPackage{%
3982         \EnableBabelHook{babel-bidi}%
3983         \bbl@loadfontspec % bidi needs fontspec
3984         \usepackage#1{bidi}%
3985         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3986         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3987           \ifnum\@nameuse{bbl@wdir@}\language\name=\tw@ % 'AL' bidi
3988             \bbl@digitsdotdash % So ignore in 'R' bidi
3989           \fi}}%
3990     \fi}
3991   \ifnum\bbl@bidimode>200 % Any xe bidi=
3992     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3993       \bbl@tentative{bidi=bidi}
3994       \bbl@loadxebidi{}
3995     \or
3996       \bbl@loadxebidi{[rldocument]}
3997     \or
3998       \bbl@loadxebidi{}
3999     \fi
4000   \fi
4001 \fi
4002 % TODO? Separate:
4003 \ifnum\bbl@bidimode=\@ne % bidi=default
4004   \let\bbl@beforeforeign\leavevmode
4005   \ifodd\bbl@engine % lua
4006     \newattribute\bbl@attr@dir
4007     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4008     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4009   \fi
4010   \AtEndOfPackage{%
4011     \EnableBabelHook{babel-bidi}% pdf/lua/xe
4012     \ifodd\bbl@engine\else % pdf/xe
4013       \bbl@xebidipar
4014     \fi}
4015 \fi

```

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

```

4016 \bbl@trace{Macros to switch the text direction}
4017 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4018 \def\bbl@rscripts{% TODO. Base on codes ??
4019   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4020   Old Hungarian,Lydian,Mandaean,Manichaeen,%
4021   Meroitic Cursive,Meroitic,Old North Arabian,%
4022   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4023   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4024   Old South Arabian,}%
4025 \def\bbl@provide@dirs#1{%
4026   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4027   \ifin@
4028     \global\bbl@csarg\chardef{wdir@#1}\@ne
4029     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4030     \ifin@
4031       \global\bbl@csarg\chardef{wdir@#1}\tw@
4032     \fi
4033   \else
4034     \global\bbl@csarg\chardef{wdir@#1}\z@
4035   \fi
4036   \ifodd\bbl@engine
4037     \bbl@csarg\ifcase{wdir@#1}%
4038       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4039     \or
4040       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4041     \or
4042       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4043     \fi
4044   \fi}
4045 \def\bbl@switchdir{%
4046   \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4047   \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4048   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4049 \def\bbl@setdirs#1{% TODO - math
4050   \ifcase\bbl@select@type % TODO - strictly, not the right test
4051     \bbl@bodydir{#1}%
4052     \bbl@pardir{#1}% <- Must precede \bbl@texdir
4053   \fi
4054   \bbl@texdir{#1}}
4055 % TODO. Only if \bbl@bidimode > 0?:
4056 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4057 \DisableBabelHook{babel-bidi}

```

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

```

4058 \ifodd\bbl@engine % luatex=1
4059 \else % pdftex=0, xetex=2
4060   \newcount\bbl@dirlevel
4061   \chardef\bbl@thetexdir\z@
4062   \chardef\bbl@thepardir\z@
4063   \def\bbl@texdir#1{%
4064     \ifcase#1\relax
4065       \chardef\bbl@thetexdir\z@
4066       \@nameuse{setlatin}%
4067       \bbl@texdir@i\beginL\endL
4068     \else
4069       \chardef\bbl@thetexdir\@ne
4070       \@nameuse{setnonlatin}%
4071       \bbl@texdir@i\beginR\endR
4072     \fi}
4073   \def\bbl@texdir@i#1#2{%
4074     \ifhmode

```



```

4075 \ifnum\currentgrouplevel>\z@
4076 \ifnum\currentgrouplevel=\bbl@dirlevel
4077 \bbl@error{multiple-bidi}{\}\}\}%
4078 \bgroup\aftergroup#2\aftergroup\egroup
4079 \else
4080 \ifcase\currentgrouptype\or % 0 bottom
4081 \aftergroup#2% 1 simple {}
4082 \or
4083 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4084 \or
4085 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4086 \or\or\or % vbox vtop align
4087 \or
4088 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4089 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4090 \or
4091 \aftergroup#2% 14 \begingroup
4092 \else
4093 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4094 \fi
4095 \fi
4096 \bbl@dirlevel\currentgrouplevel
4097 \fi
4098 #1%
4099 \fi}
4100 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4101 \let\bbl@bodydir\@gobble
4102 \let\bbl@pagedir\@gobble
4103 \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).

```

4104 \def\bbl@xebidipar{%
4105 \let\bbl@xebidipar\relax
4106 \TeXeTstate\@ne
4107 \def\bbl@xeeverypar{%
4108 \ifcase\bbl@thepardir
4109 \ifcase\bbl@thetextdir\else\beginR\fi
4110 \else
4111 {\setbox\z@\lastbox\beginR\box\z@}%
4112 \fi}%
4113 \let\bbl@severypar\everypar
4114 \newtoks\everypar
4115 \everypar=\bbl@severypar
4116 \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4117 \ifnum\bbl@bidimode>200 % Any xe bidi=
4118 \let\bbl@textdir\i\@gobbletwo
4119 \let\bbl@xebidipar\@empty
4120 \AddBabelHook{bidi}{foreign}{%
4121 \def\bbl@tempa{\def\BabelText###1}%
4122 \ifcase\bbl@thetextdir
4123 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4124 \else
4125 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4126 \fi}
4127 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4128 \fi
4129 \fi

```

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

```

4130 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4131 \AtBeginDocument{%
4132 \ifx\pdfstringdefDisableCommands\@undefined\else

```

```

4133 \ifx\pdfstringdefDisableCommands\relax\else
4134 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4135 \fi
4136 \fi}

```

## 5.6 Local Language Configuration

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

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

```

4137 \bbl@trace{Local Language Configuration}
4138 \ifx\loadlocalcfg\undefined
4139 \@ifpackagewith{babel}{noconfigs}%
4140 {\let\loadlocalcfg\@gobble}%
4141 {\def\loadlocalcfg#1{%
4142 \InputIfFileExists{#1.cfg}%
4143 {\typeout{*****^J%
4144 * Local config file #1.cfg used^^J%
4145 *}}}%
4146 \@empty}}
4147 \fi

```

## 5.7 Language options

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

```

4148 \bbl@trace{Language options}
4149 \let\bbl@afterlang\relax
4150 \let\BabelModifiers\relax
4151 \let\bbl@loaded\@empty
4152 \def\bbl@load@language#1{%
4153 \InputIfFileExists{#1.ldf}%
4154 {\edef\bbl@loaded{\CurrentOption
4155 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4156 \expandafter\let\expandafter\bbl@afterlang
4157 \csname\CurrentOption.ldf-h@k\endcsname
4158 \expandafter\let\expandafter\BabelModifiers
4159 \csname bbl@mod@\CurrentOption\endcsname
4160 \bbl@exp{\AtBeginDocument{%
4161 \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4162 {\IfFileExists{babel-#1.tex}%
4163 {\def\bbl@tempa{%
4164 .\There is a locale ini file for this language.\\%
4165 If it's the main language, try adding `provide=*'\%
4166 to the babel package options}}%
4167 {\let\bbl@tempa\empty}%
4168 \bbl@error{unknown-package-option}{\CurrentOption}}}

```

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.

```

4169 \def\bbl@try@load@lang#1#2#3{%
4170 \IfFileExists{\CurrentOption.ldf}%
4171 {\bbl@load@language{\CurrentOption}}%
4172 {\#1\bbl@load@language{#2#3}}
4173 %
4174 \DeclareOption{hebrew}{%
4175 \ifcase\bbl@engine\or
4176 \bbl@error{only-pdftex-lang}{hebrew}{\luatex}}}%

```

```

4177 \fi
4178 \input{rlbabel.def}%
4179 \bbl@load@language{hebrew}}
4180 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4181 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4182 \DeclareOption{polutonikogreek}{%
4183   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4184 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4185 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4186 \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.

```

4187 \ifx\bbl@opt@config\@nnil
4188   \@ifpackagewith{babel}{noconfigs}}}%
4189   {\InputIfFileExists{bblopts.cfg}%
4190     {\typeout{*****^J%
4191               * Local config file bblopts.cfg used^^J%
4192               *}}}%
4193     {}}}%
4194 \else
4195   \InputIfFileExists{\bbl@opt@config.cfg}%
4196     {\typeout{*****^J%
4197               * Local config file \bbl@opt@config.cfg used^^J%
4198               *}}}%
4199     {\bbl@error{config-not-found}{}}}%
4200 \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.

```

4201 \ifx\bbl@opt@main\@nnil
4202   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4203     \let\bbl@tempb\@empty
4204     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4205     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}}%
4206     \bbl@foreach\bbl@tempb{%   \bbl@tempb is a reversed list
4207       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4208         \ifodd\bbl@iniflag % = *=
4209           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4210         \else % n +=
4211           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}}%
4212       \fi
4213     \fi}%
4214 \fi
4215 \else
4216   \bbl@info{Main language set with 'main='. Except if you have\\%
4217     problems, prefer the default mechanism for setting\\%
4218     the main language, ie, as the last declared.\\%
4219     Reported}%
4220 \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`).

```

4221 \ifx\bbl@opt@main\@nnil\else
4222   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4223   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4224 \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.

```

4225 \bbl@foreach\bbl@language@opts{%
4226   \def\bbl@tempa{#1}%
4227   \ifx\bbl@tempa\bbl@opt@main\else
4228     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4229       \bbl@ifunset{ds@#1}%
4230       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4231       {}%
4232     \else % + * (other = ini)
4233       \DeclareOption{#1}{%
4234         \bbl@ldfinit
4235         \babelprovide[import]{#1}%
4236         \bbl@afterldf{}}%
4237     \fi
4238   \fi}
4239 \bbl@foreach\@classoptionslist{%
4240   \def\bbl@tempa{#1}%
4241   \ifx\bbl@tempa\bbl@opt@main\else
4242     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4243       \bbl@ifunset{ds@#1}%
4244       {\IfFileExists{#1.ldf}%
4245        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4246        {}}%
4247       {}%
4248     \else % + * (other = ini)
4249       \IfFileExists{babel-#1.tex}%
4250       {\DeclareOption{#1}{%
4251         \bbl@ldfinit
4252         \babelprovide[import]{#1}%
4253         \bbl@afterldf{}}}%
4254       {}%
4255     \fi
4256   \fi}

```

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

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

```

4257 \def\AfterBabelLanguage#1{%
4258   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4259   \DeclareOption*{}
4260   \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.

```

4261 \bbl@trace{Option 'main'}
4262 \ifx\bbl@opt@main\@nnil
4263   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4264   \let\bbl@tempc\@empty
4265   \edef\bbl@templ{\bbl@loaded,}
4266   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4267   \bbl@for\bbl@tempb\bbl@tempa{%
4268     \edef\bbl@tempd{\bbl@tempb,%}
4269     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4270     \bbl@xin{\bbl@tempd}{\bbl@templ}%
4271     \ifin\edef\bbl@tempc{\bbl@tempb}\fi
4272   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4273   \expandafter\bbl@tempa\bbl@loaded,\@nnil

```

```

4274 \ifx\bbl@tempb\bbl@tempc\else
4275   \bbl@warning{%
4276     Last declared language option is '\bbl@tempc',\%
4277     but the last processed one was '\bbl@tempb'.\%
4278     The main language can't be set as both a global\%
4279     and a package option. Use 'main=\bbl@tempc' as\%
4280     option. Reported}
4281 \fi
4282 \else
4283 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4284   \bbl@ldfinit
4285   \let\CurrentOption\bbl@opt@main
4286   \bbl@exp{% \bbl@opt@provide = empty if *
4287     \\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4288   \bbl@afterldf{}
4289   \DeclareOption{\bbl@opt@main}{}
4290 \else % case 0,2 (main is ldf)
4291   \ifx\bbl@loadmain\relax
4292     \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4293   \else
4294     \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4295   \fi
4296   \ExecuteOptions{\bbl@opt@main}
4297   \@namedef{ds@\bbl@opt@main}{}%
4298 \fi
4299 \DeclareOption*{}
4300 \ProcessOptions*
4301 \fi
4302 \bbl@exp{%
4303   \\AtBeginDocument{\\bbl@usehooks@lang{/}{begindocument}{}}}%
4304 \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.

```

4305 \ifx\bbl@main@language\@undefined
4306   \bbl@info{%
4307     You haven't specified a language as a class or package\%
4308     option. I'll load 'nil'. Reported}
4309   \bbl@load@language{nil}
4310 \fi
4311 \</package>

```

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

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

Because plain  $\text{T}_{\text{E}}\text{X}$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\text{T}_{\text{E}}\text{X}$  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  $\text{T}_{\text{E}}\text{X}$  and  $\text{\LaTeX}$ , some of it is for the  $\text{\LaTeX}$  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`

```

4312 \<*kernel>
4313 \let\bbl@onlyswitch\@empty
4314 \input babel.def
4315 \let\bbl@onlyswitch\@undefined
4316 \</kernel>
4317 %
4318 % \section{Error messages}

```

```

4319%
4320% They are loaded when |\bbl@error| is first called. To save space, the
4321% main code just identifies them with a tag, and messages are stored in
4322% a separate file. Since it can be loaded anywhere, you make sure some
4323% catcodes have the right value, although those for |\|, |`|, |^M|,
4324% |%| and |=| are reset before loading the file.
4325%
4326 (*errors)
4327 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4328 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4329 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4330 \catcode`\@=11 \catcode`\^=7
4331%
4332 \ifx\MessageBreak@undefined
4333   \gdef\bbl@error@i#1#2{%
4334     \begingroup
4335       \newlinechar=`^^J
4336       \def\{^^J(babel) }%
4337       \errhelp{#2}\errmessage{\{#1}%
4338     \endgroup}
4339 \else
4340   \gdef\bbl@error@i#1#2{%
4341     \begingroup
4342       \def\{\MessageBreak}%
4343       \PackageError{babel}{#1}{#2}%
4344     \endgroup}
4345 \fi
4346 \def\bbl@errmessage#1#2#3{%
4347   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4348     \bbl@error@i{#2}{#3}}
4349 % Implicit #2#3#4:
4350 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4351%
4352 \bbl@errmessage{not-yet-available}
4353   {Not yet available}%
4354   {Find an armchair, sit down and wait}
4355 \bbl@errmessage{bad-package-option}%
4356   {Bad option '#1=#2'. Either you have misspelled the\\%
4357     key or there is a previous setting of '#1'. Valid\\%
4358     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4359     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4360   {See the manual for further details.}
4361 \bbl@errmessage{base-on-the-fly}
4362   {For a language to be defined on the fly 'base'\\%
4363     is not enough, and the whole package must be\\%
4364     loaded. Either delete the 'base' option or\\%
4365     request the languages explicitly}%
4366   {See the manual for further details.}
4367 \bbl@errmessage{undefined-language}
4368   {You haven't defined the language '#1' yet.\\%
4369     Perhaps you misspelled it or your installation\\%
4370     is not complete}%
4371   {Your command will be ignored, type <return> to proceed}
4372 \bbl@errmessage{shorthand-is-off}
4373   {I can't declare a shorthand turned off (\string#2)}
4374   {Sorry, but you can't use shorthands which have been\\%
4375     turned off in the package options}
4376 \bbl@errmessage{not-a-shorthand}
4377   {The character '\string #1' should be made a shorthand character;\\%
4378     add the command \string\usesshorthands\string{#1\string} to
4379     the preamble.\\%
4380     I will ignore your instruction}%
4381   {You may proceed, but expect unexpected results}

```

```

4382 \bbl@errmessage{not-a-shorthand-b}
4383   {I can't switch '\string#2' on or off--not a shorthand}%
4384   {This character is not a shorthand. Maybe you made\\%
4385     a typing mistake? I will ignore your instruction.}
4386 \bbl@errmessage{unknown-attribute}
4387   {The attribute #2 is unknown for language #1.}%
4388   {Your command will be ignored, type <return> to proceed}
4389 \bbl@errmessage{missing-group}
4390   {Missing group for string \string#1}%
4391   {You must assign strings to some category, typically\\%
4392     captions or extras, but you set none}
4393 \bbl@errmessage{only-lua-xe}
4394   {This macro is available only in LuaLaTeX and XeLaTeX.}%
4395   {Consider switching to these engines.}
4396 \bbl@errmessage{only-lua}
4397   {This macro is available only in LuaLaTeX.}%
4398   {Consider switching to that engine.}
4399 \bbl@errmessage{unknown-provide-key}
4400   {Unknown key '#1' in \string\babelprovide}%
4401   {See the manual for valid keys}%
4402 \bbl@errmessage{unknown-mapfont}
4403   {Option '\bbl@KVP@mapfont' unknown for\\%
4404     mapfont. Use 'direction'.}%
4405   {See the manual for details.}
4406 \bbl@errmessage{no-ini-file}
4407   {There is no ini file for the requested language\\%
4408     (#1: \languagename). Perhaps you misspelled it or your\\%
4409     installation is not complete.}%
4410   {Fix the name or reinstall babel.}
4411 \bbl@errmessage{digits-is-reserved}
4412   {The counter name 'digits' is reserved for mapping\\%
4413     decimal digits}%
4414   {Use another name.}
4415 \bbl@errmessage{limit-two-digits}
4416   {Currently two-digit years are restricted to the\\
4417     range 0-9999.}%
4418   {There is little you can do. Sorry.}
4419 \bbl@errmessage{alphabetic-too-large}
4420   {Alphabetic numeral too large (#1)}%
4421   {Currently this is the limit.}
4422 \bbl@errmessage{no-ini-info}
4423   {I've found no info for the current locale.\\%
4424     The corresponding ini file has not been loaded\\%
4425     Perhaps it doesn't exist}%
4426   {See the manual for details.}
4427 \bbl@errmessage{unknown-ini-field}
4428   {Unknown field '#1' in \string\BCPdata.\\%
4429     Perhaps you misspelled it.}%
4430   {See the manual for details.}
4431 \bbl@errmessage{unknown-locale-key}
4432   {Unknown key for locale '#2':\\%
4433     #3\\%
4434     \string#1 will be set to \relax}%
4435   {Perhaps you misspelled it.}%
4436 \bbl@errmessage{adjust-only-vertical}
4437   {Currently, #1 related features can be adjusted only\\%
4438     in the main vertical list.}%
4439   {Maybe things change in the future, but this is what it is.}
4440 \bbl@errmessage{layout-only-vertical}
4441   {Currently, layout related features can be adjusted only\\%
4442     in vertical mode.}%
4443   {Maybe things change in the future, but this is what it is.}
4444 \bbl@errmessage{bidi-only-lua}

```

```

4445 {The bidi method 'basic' is available only in\\%
4446 luatex. I'll continue with 'bidi=default', so\\%
4447 expect wrong results}%
4448 {See the manual for further details.}
4449 \bbl@errmessage{multiple-bidi}
4450 {Multiple bidi settings inside a group}%
4451 {I'll insert a new group, but expect wrong results.}
4452 \bbl@errmessage{unknown-package-option}
4453 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4454 or the language definition file \CurrentOption.ldf\\%
4455 was not found%
4456 \bbl@tempa}
4457 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4458 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4459 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4460 \bbl@errmessage{config-not-found}
4461 {Local config file '\bbl@opt@config.cfg' not found}%
4462 {Perhaps you misspelled it.}
4463 \bbl@errmessage{late-after-babel}
4464 {Too late for \string\AfterBabelLanguage}%
4465 {Languages have been loaded, so I can do nothing}
4466 \bbl@errmessage{double-hyphens-class}
4467 {Double hyphens aren't allowed in \string\babelcharclass\\%
4468 because it's potentially ambiguous}%
4469 {See the manual for further info}
4470 \bbl@errmessage{unknown-interchar}
4471 {'#1' for '\language' cannot be enabled.\\%
4472 Maybe there is a typo.}%
4473 {See the manual for further details.}
4474 \bbl@errmessage{unknown-interchar-b}
4475 {'#1' for '\language' cannot be disabled.\\%
4476 Maybe there is a typo.}%
4477 {See the manual for further details.}
4478 \bbl@errmessage{charproperty-only-vertical}
4479 {\string\babelcharproperty\space can be used only in\\%
4480 vertical mode (preamble or between paragraphs)}%
4481 {See the manual for further info}
4482 \bbl@errmessage{unknown-char-property}
4483 {No property named '#2'. Allowed values are\\%
4484 direction (bc), mirror (bmg), and linebreak (lb)}%
4485 {See the manual for further info}
4486 \bbl@errmessage{bad-transform-option}
4487 {Bad option '#1' in a transform.\\%
4488 I'll ignore it but expect more errors}%
4489 {See the manual for further info.}
4490 \bbl@errmessage{font-conflict-transforms}
4491 {Transforms cannot be re-assigned to different\\%
4492 fonts. The conflict is in '\bbl@kv@label'.\\%
4493 Apply the same fonts or use a different label}%
4494 {See the manual for further details.}
4495 \bbl@errmessage{transform-not-available}
4496 {'#1' for '\language' cannot be enabled.\\%
4497 Maybe there is a typo or it's a font-dependent transform}%
4498 {See the manual for further details.}
4499 \bbl@errmessage{transform-not-available-b}
4500 {'#1' for '\language' cannot be disabled.\\%
4501 Maybe there is a typo or it's a font-dependent transform}%
4502 {See the manual for further details.}
4503 \bbl@errmessage{year-out-range}
4504 {Year out of range.\\%
4505 The allowed range is #1}%
4506 {See the manual for further details.}
4507 \bbl@errmessage{only-pdfTeX-lang}

```



```

4508 {The '#1' ldf style doesn't work with #2,\\%
4509 but you can use the ini locale instead.\\%
4510 Try adding 'provide=*' to the option list. You may\\%
4511 also want to set 'bidi=' to some value.}%
4512 {See the manual for further details.}
4513 </errors>
4514 <*patterns>

```

## 7 Loading hyphenation patterns

The following code is meant to be read by  $\text{\texttt{iniTeX}}$  because it should instruct  $\text{\texttt{TeX}}$  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.

```

4515 <<Make sure ProvidesFile is defined>>
4516 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4517 \xdef\bbl@format{\jobname}
4518 \def\bbl@version{<<version>>}
4519 \def\bbl@date{<<date>>}
4520 \ifx\AtBeginDocument\undefined
4521 \def\@empty{}
4522 \fi
4523 <<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.

```

4524 \def\process@line#1#2 #3 #4 {%
4525 \ifx=#1%
4526 \process@synonym{#2}%
4527 \else
4528 \process@language{#1#2}{#3}{#4}%
4529 \fi
4530 \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.

```

4531 \toks@{}
4532 \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 `hyphenmins` parameters for the synonym.

```

4533 \def\process@synonym#1{%
4534 \ifnum\last@language=\m@ne
4535 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4536 \else
4537 \expandafter\chardef\csname l@#1\endcsname\last@language
4538 \wlog{\string\l@#1=\string\language\the\last@language}%
4539 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4540 \csname\language\hyphenmins\endcsname
4541 \let\bbl@elt\relax
4542 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4543 \fi}

```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```

4544 \def\process@language#1#2#3{%
4545   \expandafter\addlanguage\csname l@#1\endcsname
4546   \expandafter\language\csname l@#1\endcsname
4547   \edef\language#1#2#3{%
4548     \bbl@hook@everylanguage{#1}%
4549     % > luatex
4550     \bbl@get@enc#1:.\@@@
4551     \begingroup
4552       \lefthyphenmin\m@ne
4553       \bbl@hook@loadpatterns{#2}%
4554       % > luatex
4555       \ifnum\lefthyphenmin=\m@ne
4556       \else
4557         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4558           \the\lefthyphenmin\the\righthyphenmin}%
4559       \fi
4560     \endgroup
4561     \def\bbl@tempa{#3}%
4562     \ifx\bbl@tempa\empty\else
4563       \bbl@hook@loadexceptions{#3}%
4564       % > luatex
4565     \fi
4566     \let\bbl@elt\relax
4567     \edef\bbl@languages{%
4568       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4569     \ifnum\the\language=\z@
4570       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4571         \set@hyphenmins\tw@\thr@@\relax
4572       \else
4573         \expandafter\expandafter\expandafter\set@hyphenmins
4574         \csname #1hyphenmins\endcsname
4575       \fi
4576       \the\toks@
4577       \toks@{}%
4578     \fi}

```

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

```

4579 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides `luatex`, format-specific configuration files are taken into account. `loadkernel` currently loads nothing, but

define some basic macros instead.

```
4580 \def\bbl@hook@everylanguage#1{}
4581 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4582 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4583 \def\bbl@hook@loadkernel#1{%
4584   \def\addlanguage{\csname newlanguage\endcsname}%
4585   \def\adddialect##1##2{%
4586     \global\chardef##1##2\relax
4587     \wlog{\string##1 = a dialect from \string\language##2}}%
4588   \def\iflanguage##1{%
4589     \expandafter\ifx\csname l@##1\endcsname\relax
4590       \nolanner{##1}%
4591     \else
4592       \ifnum\csname l@##1\endcsname=\language
4593         \expandafter\expandafter\expandafter\@firstoftwo
4594       \else
4595         \expandafter\expandafter\expandafter\@secondoftwo
4596       \fi
4597     \fi}%
4598   \def\providehyphenmins##1##2{%
4599     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4600       \namedef{##1hyphenmins}{##2}%
4601     \fi}%
4602   \def\set@hyphenmins##1##2{%
4603     \lefthyphenmin##1\relax
4604     \righthyphenmin##2\relax}%
4605   \def\selectlanguage{%
4606     \errhelp{Selecting a language requires a package supporting it}%
4607     \errmessage{Not loaded}}%
4608   \let\foreignlanguage\selectlanguage
4609   \let\otherlanguage\selectlanguage
4610   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4611   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4612     \def\setlocale{%
4613       \errhelp{Find an armchair, sit down and wait}%
4614       \errmessage{(babel) Not yet available}}%
4615     \let\uselocale\setlocale
4616     \let\locale\setlocale
4617     \let\selectlocale\setlocale
4618     \let\localename\setlocale
4619     \let\textlocale\setlocale
4620     \let\textlanguage\setlocale
4621     \let\languagetext\setlocale}
4622   \begingroup
4623     \def\AddBabelHook#1#2{%
4624       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4625         \def\next{\toks1}%
4626       \else
4627         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4628       \fi
4629       \next}
4630   \ifx\directlua\@undefined
4631     \ifx\XeTeXinputencoding\@undefined\else
4632       \input xebabel.def
4633     \fi
4634   \else
4635     \input luababel.def
4636   \fi
4637   \openin1 = babel-\bbl@format.cfg
4638   \ifeof1
4639   \else
4640     \input babel-\bbl@format.cfg\relax
4641   \fi
```

```

4642 \closein1
4643 \endgroup
4644 \bbl@hook@loadkernel{switch.def}

```

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

```

4645 \openin1 = language.dat

```

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

```

4646 \def\language{english}%
4647 \ifeof1
4648 \message{I couldn't find the file language.dat,\space
4649         I will try the file hyphen.tex}
4650 \input hyphen.tex\relax
4651 \chardef\l@english\z@
4652 \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.

```

4653 \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.

```

4654 \loop
4655   \endlinechar\m@ne
4656   \read1 to \bbl@line
4657   \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.

```

4658   \if T\ifeof1F\fi T\relax
4659   \ifx\bbl@line\@empty\else
4660     \edef\bbl@line{\bbl@line\space\space\space}%
4661     \expandafter\process@line\bbl@line\relax
4662   \fi
4663 \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.

```

4664 \begingroup
4665   \def\bbl@elt#1#2#3#4{%
4666     \global\language=#2\relax
4667     \gdef\language{#1}%
4668     \def\bbl@elt##1##2##3##4{}}%
4669   \bbl@languages
4670 \endgroup
4671 \fi
4672 \closein1

```

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

```

4673 \if/\the\toks@/\else
4674   \errhelp{language.dat loads no language, only synonyms}
4675   \errmessage{Orphan language synonym}
4676 \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.

```

4677 \let\bbl@line\@undefined
4678 \let\process@line\@undefined

```

```

4679 \let\process@synonym\@undefined
4680 \let\process@language\@undefined
4681 \let\bbl@get@enc\@undefined
4682 \let\bbl@hyph@enc\@undefined
4683 \let\bbl@tempa\@undefined
4684 \let\bbl@hook@loadkernel\@undefined
4685 \let\bbl@hook@everylanguage\@undefined
4686 \let\bbl@hook@loadpatterns\@undefined
4687 \let\bbl@hook@loadexceptions\@undefined
4688 \</patterns>

```

Here the code for `iniTeX` ends.

## 8 Font handling with fontspec

Add the bidi handler just before `luaotfload`, which is loaded by default by `LaTeX`. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```

4689 <<More package options>> ≡
4690 \chardef\bbl@bidimode\z@
4691 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4692 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4693 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4694 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4695 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4696 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4697 <</More package options>>

```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\.family` by the corresponding macro `\.default`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to `babel`, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading (and mostly unuseful) message.

```

4698 <<Font selection>> ≡
4699 \bbl@trace{Font handling with fontspec}
4700 \ifx\ExplSyntaxOn\@undefined\else
4701   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4702     \in@{,#1,}{,no-script,language-not-exist,}%
4703     \ifin@else\bbl@tempfs@nx{#1}{#2}\fi}
4704   \def\bbl@fs@warn@nxx#1#2#3{%
4705     \in@{,#1,}{,no-script,language-not-exist,}%
4706     \ifin@else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4707   \def\bbl@loadfontspec{%
4708     \let\bbl@loadfontspec\relax
4709     \ifx\fontspec\@undefined
4710       \usepackage{fontspec}%
4711     \fi}%
4712 \fi
4713 \@onlypreamble\babelfont
4714 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4715   \bbl@foreach{#1}{%
4716     \expandafter\ifx\csname date##1\endcsname\relax
4717       \IfFileExists{babel-##1.tex}%
4718       {\babelprovide{##1}}%
4719     }%
4720   \fi}%
4721 \edef\bbl@tempa{#1}%
4722 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4723 \bbl@loadfontspec
4724 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4725 \bbl@bblfont}
4726 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4727   \bbl@ifunset{\bbl@tempb family}%

```

```

4728     {\bbl@providefam{\bbl@tempb}}}%
4729     {}%
4730 % For the default font, just in case:
4731 \bbl@ifunset{\bbl@lsys{\language}\bbl@provide@lsys{\language}}{}%
4732 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4733 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}}% save bbl@rmdflt@
4734 \bbl@exp{%
4735     \let<\bbl@tempb dflt@\language><\bbl@tempb dflt@>%
4736     \bbl@font@set<\bbl@tempb dflt@\language>%
4737     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4738 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4739     \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}}}%

```

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

```

4740 \def\bbl@providefam#1{%
4741     \bbl@exp{%
4742         \\\newcommand<#1default>{}% Just define it
4743         \\\bbl@add@list\\bbl@font@fams{#1}%
4744         \\\DeclareRobustCommand<#1family>{%
4745             \\\not@math@alphabet<#1family>\relax
4746             % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4747             \\\fontfamily<#1default>%
4748             <{ifx>\\UseHooks\\<undefined><else>\\UseHook{#1family}<fi>%
4749             \\\selectfont}%
4750             \\\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.

```

4751 \def\bbl@nostdfont#1{%
4752     \bbl@ifunset{\bbl@WFF@f@family}%
4753     {\bbl@csarg\gdef{WFF@f@family}}% Flag, to avoid dupl warns
4754     \bbl@infowarn{The current font is not a babel standard family:\\%
4755         #1%
4756         \fontname\font\\%
4757         There is nothing intrinsically wrong with this warning, and\\%
4758         you can ignore it altogether if you do not need these\\%
4759         families. But if they are used in the document, you should be\\%
4760         aware 'babel' will not set Script and Language for them, so\\%
4761         you may consider defining a new family with \string\babelfont.\\%
4762         See the manual for further details about \string\babelfont.\\%
4763         Reported}}
4764     {}}%
4765 \gdef\bbl@switchfont{%
4766     \bbl@ifunset{\bbl@lsys{\language}\bbl@provide@lsys{\language}}{}%
4767     \bbl@exp{% eg Arabic -> arabic
4768         \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4769     \bbl@foreach\bbl@font@fams{%
4770         \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4771         {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4772             {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4773                 {}% 123=F - nothing!
4774                 {\bbl@exp{% 3=T - from generic
4775                     \global\let<\bbl@##1dflt@\language>%
4776                     \<\bbl@##1dflt@>}}}%
4777                 {\bbl@exp{% 2=T - from script
4778                     \global\let<\bbl@##1dflt@\language>%
4779                     \<\bbl@##1dflt@*\bbl@tempa>}}}%
4780                 {}% 1=T - language, already defined
4781         \def\bbl@tempa{\bbl@nostdfont}}% TODO. Don't use \bbl@tempa
4782     \bbl@foreach\bbl@font@fams{% don't gather with prev for
4783         \bbl@ifunset{\bbl@##1dflt@\language}%
4784         {\bbl@cs{famrst@##1}%
4785             \global\bbl@csarg\let{famrst@##1}\relax}%
4786         {\bbl@exp{% order is relevant. TODO: but sometimes wrong!

```

```

4787     \\bbl@add\\originalTeX{%
4788     \\bbl@font@rst{\\bbl@cl{##1dflt}}}%
4789     \\<##1default>\\<##1family>{##1}}%
4790     \\bbl@font@set\\<bbl@##1dflt@\\language\\>% the main part!
4791     \\<##1default>\\<##1family>}}}%
4792     \\bbl@ifrestoring{\\bbl@tempa}}}%

The following is executed at the beginning of the aux file or the document to warn about fonts not
defined with \\babelfont.

4793 \\ifx\\f@family\\undefined\\else    % if latex
4794 \\ifcase\\bbl@engine                % if pdftex
4795     \\let\\bbl@cckcckstdfonts\\relax
4796 \\else
4797     \\def\\bbl@cckcckstdfonts{%
4798     \\begingroup
4799     \\global\\let\\bbl@cckcckstdfonts\\relax
4800     \\let\\bbl@tempa\\empty
4801     \\bbl@foreach\\bbl@font@fams{%
4802     \\bbl@ifunset{\\bbl@##1dflt@}%
4803     {\\@nameuse{##1family}}%
4804     \\bbl@csarg\\gdef{WFF@\\f@family}}}% Flag
4805     \\bbl@exp{\\bbl@add\\bbl@tempa{* \\<##1family>= \\f@family\\}%
4806     \\space\\space\\fontname\\font\\}%
4807     \\bbl@csarg\\xdef{##1dflt@}{\\f@family}%
4808     \\expandafter\\xdef\\csname ##1default\\endcsname{\\f@family}}%
4809     }%
4810     \\ifx\\bbl@tempa\\empty\\else
4811     \\bbl@infowarn{The following font families will use the default\\%
4812     settings for all or some languages:\\%
4813     \\bbl@tempa
4814     There is nothing intrinsically wrong with it, but\\%
4815     'babel' will no set Script and Language, which could\\%
4816     be relevant in some languages. If your document uses\\%
4817     these families, consider redefining them with \\string\\babelfont.\\%
4818     Reported}%
4819     \\fi
4820     \\endgroup}
4821 \\fi
4822 \\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,  $\TeX$  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 >ssub\*).

```

4823 \\def\\bbl@font@set#1#2#3{% eg \\bbl@rmdflt@lang \\rmdefault \\rmfamily
4824     \\bbl@xin@{<>}{#1}%
4825     \\ifin@
4826     \\bbl@exp{\\bbl@fontspec@set\\#1\\expandafter\\@gobbletwo#1\\#3}%
4827     \\fi
4828     \\bbl@exp{%
4829     \\def\\#2{#1}% eg, \\rmdefault{\\bbl@rmdflt@lang}
4830     \\bbl@ifsamestring{#2}{\\f@family}%
4831     {\\#3%
4832     \\bbl@ifsamestring{\\f@series}{\\bfdefault}{\\bfseries}}}%
4833     \\let\\bbl@tempa\\relax}%
4834     }%
4835 % TODO - next should be global?, but even local does its job. I'm
4836 % still not sure -- must investigate:

```

```

4837 \def\bbf@fontspec@set#1#2#3#4{% eg \bbf@rmdflt@lang fnt-opt fnt-nme \xxfamily
4838 \let\bbf@tempe\bbf@mapselect
4839 \edef\bbf@tempb{\bbf@stripslash#4/}% Catcodes hack (better pass it).
4840 \bbf@exp{\bbf@replace{\bbf@tempb{\bbf@stripslash\family/}}}%
4841 \let\bbf@mapselect\relax
4842 \let\bbf@temp@fam#4% eg, '\rmfamily', to be restored below
4843 \let#4\empty % Make sure \renewfontfamily is valid
4844 \bbf@exp{%
4845 \let\bbf@temp@pfam<\bbf@stripslash#4\space>% eg, '\rmfamily '
4846 \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbf@cl{sname}}}%
4847 {\newfontscript{\bbf@cl{sname}}{\bbf@cl{sotf}}}%
4848 \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbf@cl{lname}}}%
4849 {\newfontlanguage{\bbf@cl{lname}}{\bbf@cl{lotf}}}%
4850 \let\bbf@tempfs@nx<__fontspec_warning:nx>%
4851 \let<__fontspec_warning:nx>\bbf@fs@warn@nx
4852 \let\bbf@tempfs@nxx<__fontspec_warning:nxx>%
4853 \let<__fontspec_warning:nxx>\bbf@fs@warn@nxx
4854 \renewfontfamily\#4%
4855 [\bbf@cl{sys},% xetex removes unknown features :-(
4856 \ifcase\bbf@engine\or RawFeature={family=\bbf@tempb},\fi
4857 #2}{#3}% ie \bbf@exp{..}{#3}
4858 \bbf@exp{%
4859 \let<__fontspec_warning:nx>\bbf@tempfs@nx
4860 \let<__fontspec_warning:nxx>\bbf@tempfs@nxx}%
4861 \begingroup
4862 #4%
4863 \xdef#1{\f@family}% eg, \bbf@rmdflt@lang{FreeSerif(0)}
4864 \endgroup % TODO. Find better tests:
4865 \bbf@xin@{\string>\string s\string s\string u\string b\string*}%
4866 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4867 \ifin@
4868 \global\bbf@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4869 \fi
4870 \bbf@xin@{\string>\string s\string s\string u\string b\string*}%
4871 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4872 \ifin@
4873 \global\bbf@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4874 \fi
4875 \let#4\bbf@temp@fam
4876 \bbf@exp{\let<\bbf@stripslash#4\space>\bbf@temp@pfam
4877 \let\bbf@mapselect\bbf@tempe}%

```

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

```

4878 \def\bbf@font@rst#1#2#3#4{%
4879 \bbf@csarg\def{famrst@#4}{\bbf@font@set{#1}#2#3}}

```

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

```

4880 \def\bbf@font@fams{rm,sf,tt}
4881 <</Font selection>>

```

## 9 Hooks for XeTeX and LuaTeX

### 9.1 XeTeX

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

```

4882 <<(*Footnote changes)>> ≡
4883 \bbf@trace{Bidi footnotes}
4884 \ifnum\bbf@bidimode>\z@ % Any bidi=
4885 \def\bbf@footnote#1#2#3{%
4886 \ifnextchar{%

```



```

4887     {\bbl@footnote@o{#1}{#2}{#3}}%
4888     {\bbl@footnote@x{#1}{#2}{#3}}}
4889 \long\def\bbl@footnote@x#1#2#3#4{%
4890   \bgroup
4891   \select@language@x{\bbl@main@language}%
4892   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4893   \egroup}
4894 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4895   \bgroup
4896   \select@language@x{\bbl@main@language}%
4897   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4898   \egroup}
4899 \def\bbl@footnotetext#1#2#3{%
4900   \@ifnextchar{%
4901     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4902     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4903 \long\def\bbl@footnotetext@x#1#2#3#4{%
4904   \bgroup
4905   \select@language@x{\bbl@main@language}%
4906   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4907   \egroup}
4908 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4909   \bgroup
4910   \select@language@x{\bbl@main@language}%
4911   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4912   \egroup}
4913 \def\BabelFootnote#1#2#3#4{%
4914   \ifx\bbl@fn@footnote\@undefined
4915     \let\bbl@fn@footnote\footnote
4916   \fi
4917   \ifx\bbl@fn@footnotetext\@undefined
4918     \let\bbl@fn@footnotetext\footnotetext
4919   \fi
4920   \bbl@ifblank{#2}%
4921   {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4922    \@namedef{\bbl@stripslash#1text}%
4923    {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4924   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4925    \@namedef{\bbl@stripslash#1text}%
4926    {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4927 \fi
4928 <</Footnote changes>>

```

Now, the code.

```

4929 (*xetex)
4930 \def\BabelStringsDefault{unicode}
4931 \let\xebbl@stop\relax
4932 \AddBabelHook{xetex}{encodedcommands}{%
4933   \def\bbl@tempa{#1}%
4934   \ifx\bbl@tempa\@empty
4935     \XeTeXinputencoding"bytes"%
4936   \else
4937     \XeTeXinputencoding"#1"%
4938   \fi
4939   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4940 \AddBabelHook{xetex}{stopcommands}{%
4941   \xebbl@stop
4942   \let\xebbl@stop\relax}
4943 \def\bbl@input@classes{% Used in CJK intraspaces
4944   \input{load-unicode-xetex-classes.tex}%
4945   \let\bbl@input@classes\relax}
4946 \def\bbl@intraspace#1 #2 #3\@@{%
4947   \bbl@csarg\gdef{xeisp@language#1}%

```

```

4948     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4949 \def\bbl@intrapenalty#1\@{
4950   \bbl@csarg\gdef{xeipn@\language\name}%
4951   {\XeTeXlinebreakpenalty #1\relax}}
4952 \def\bbl@provide@intraspace{%
4953   \bbl@xin@{/s}{/\bbl@ccl{lnbrk}}}%
4954   \ifin@ \else \bbl@xin@{/c}{/\bbl@ccl{lnbrk}} \fi
4955   \ifin@
4956     \bbl@ifunset{\bbl@intsp@\language\name}{}%
4957     {\expandafter\ifx\csname bbl@intsp@\language\name\endcsname\@empty\else
4958       \ifx\bbl@KVP@intraspace\@nnil
4959         \bbl@exp{%
4960           \\bbl@intraspace\bbl@ccl{intsp}\\\@}%
4961         \fi
4962         \ifx\bbl@KVP@intrapenalty\@nnil
4963           \bbl@intrapenalty0\@@
4964         \fi
4965       \fi
4966       \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4967         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4968       \fi
4969       \ifx\bbl@KVP@intrapenalty\@nnil\else
4970         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4971       \fi
4972       \bbl@exp{%
4973         % TODO. Execute only once (but redundant):
4974         \\bbl@add\<extras\language\name>%
4975         \XeTeXlinebreaklocale "\bbl@ccl{tbcpr}"%
4976         \<bbl@xeisp@\language\name>%
4977         \<bbl@xeipn@\language\name>}%
4978         \\bbl@tglobal\<extras\language\name>%
4979         \\bbl@add\<noextras\language\name>%
4980         \XeTeXlinebreaklocale ""}%
4981         \\bbl@tglobal\<noextras\language\name>}%
4982       \ifx\bbl@ispacesize\@undefined
4983         \gdef\bbl@ispacesize{\bbl@ccl{xeisp}}}%
4984       \ifx\AtBeginDocument\@notprerr
4985         \expandafter\@secondoftwo % to execute right now
4986       \fi
4987       \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4988     \fi}%
4989 \fi}
4990 \ifx\DisableBabelHook\@undefined\endinput\fi %%% TODO: why
4991 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4992 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4993 \DisableBabelHook{babel-fontspec}
4994 \langle Font selection \rangle
4995 \def\bbl@provide@extra#1{}

```

## 10 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.

```

4996 \ifnum\Xe@alloc@intercharclass<\thr@@
4997   \Xe@alloc@intercharclass\thr@@
4998 \fi
4999 \chardef\bbl@xeiclass@default=\z@
5000 \chardef\bbl@xeiclass@cjkideogram=\@ne
5001 \chardef\bbl@xeiclass@cjkleftpunctuation=\tw@
5002 \chardef\bbl@xeiclass@cjkrightpunctuation=\thr@@
5003 \chardef\bbl@xeiclass@boundary=4095
5004 \chardef\bbl@xeiclass@ignore=4096

```

The machinery is activated with a hook (enabled only if actually used). Here `\bbl@tempc` is pre-set with `\bbl@usingxeclass`, 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.

```

5005 \AddBabelHook{babel-interchar}{beforeextras}{%
5006   \@nameuse{bbl@xechars@\language\language}}
5007 \DisableBabelHook{babel-interchar}
5008 \protected\def\bbl@charclass#1{%
5009   \ifnum\count@<\z@
5010     \count@-\count@
5011     \loop
5012       \bbl@exp{%
5013         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5014         \XeTeXcharclass\count@ \bbl@tempc
5015         \ifnum\count@<`#1\relax
5016         \advance\count@\@ne
5017       \repeat
5018   \else
5019     \babel@savevariable{\XeTeXcharclass`#1}%
5020     \XeTeXcharclass`#1 \bbl@tempc
5021   \fi
5022   \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@usingxeclass\bbl@xeclass@punct@english\bbl@charclass{.}` `\bbl@charclass{,}` (etc.), where `\bbl@usingxeclass` stores the class to be applied to the subsequent characters. The `\ifcat` part deals with the alternative way to enter characters as macros (eg, `\j`). As a special case, hyphens are stored as `\bbl@upto`, to deal with ranges.

```

5023 \newcommand\bbl@ifinterchar[1]{%
5024   \let\bbl@tempa\@gobble           % Assume to ignore
5025   \edef\bbl@tempb{\zap@space#1 \@empty}%
5026   \ifx\bbl@KVP@interchar\@nnil\else
5027     \bbl@replace\bbl@KVP@interchar{ }{,}%
5028     \bbl@foreach\bbl@tempb{%
5029       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5030       \ifin@
5031         \let\bbl@tempa\@firstofone
5032       \fi}%
5033   \fi
5034   \bbl@tempa}
5035 \newcommand\IfBabelIntercharT[2]{%
5036   \bbl@carg\bbl@add{\bbl@icsave@\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5037 \newcommand\babelcharclass[3]{%
5038   \EnableBabelHook{babel-interchar}%
5039   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5040   \def\bbl@tempb##1{%
5041     \ifx##1\@empty\else
5042       \ifx##1-%
5043         \bbl@upto
5044       \else
5045         \bbl@charclass{%
5046           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5047       \fi
5048       \expandafter\bbl@tempb
5049     \fi}%
5050   \bbl@ifunset{\bbl@xechars@#1}%
5051   {\toks@{%
5052     \babel@savevariable\XeTeXinterchartokenstate
5053     \XeTeXinterchartokenstate\@ne
5054   }}%
5055   {\toks@\expandafter\expandafter\expandafter{%
5056     \csname bbl@xechars@#1\endcsname}}}%

```

```

5057 \bbl@csarg\edef{xechars@#1}{%
5058   \the\toks@
5059   \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5060   \bbl@tempb#3\@empty}}
5061 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
5062 \protected\def\bbl@upto{%
5063   \ifnum\count@>\z@
5064     \advance\count@\@ne
5065     \count@-\count@
5066   \else\ifnum\count@=\z@
5067     \bbl@charclass{-}%
5068   \else
5069     \bbl@error{double-hyphens-class}{\count@}{\count@}%
5070   \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 a intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<lang>`.

```

5071 \def\bbl@ignoreinterchar{%
5072   \ifnum\language=\l@nohyphenation
5073     \expandafter\@gobble
5074   \else
5075     \expandafter\@firstofone
5076   \fi}
5077 \newcommand\babelinterchar[5][{}]{%
5078   \let\bbl@kv@label\@empty
5079   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5080   \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5081   {\bbl@ignoreinterchar{#5}}%
5082   \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5083   \bbl@exp{\bbl@for\bbl@tempa{\zap@space#3 \@empty}}%
5084   \bbl@exp{\bbl@for\bbl@tempb{\zap@space#4 \@empty}}%
5085   \XeTeXinterchartoks
5086     \@nameuse{bbl@xeclass@\bbl@tempa @#2}%
5087     \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{\bbl@tempa @#2} %
5088     \@nameuse{bbl@xeclass@\bbl@tempb @#2}%
5089     \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{\bbl@tempb @#2} %
5090     = \expandafter{%
5091       \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5092       \csname\zap@space bbl@xeinter@\bbl@kv@label
5093         @#3@#4@#2 \@empty\endcsname}}}}
5094 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5095   \bbl@ifunset{bbl@ic@#1@\language}%
5096   {\bbl@error{unknown-interchar}{#1}{\language}}%
5097   {\bbl@csarg\let{ic@#1@\language}\@firstofone}}
5098 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5099   \bbl@ifunset{bbl@ic@#1@\language}%
5100   {\bbl@error{unknown-interchar-b}{#1}{\language}}%
5101   {\bbl@csarg\let{ic@#1@\language}\@gobble}}
5102 </xetex>

```

## 10.1 Layout

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

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

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

```

5103 <*xetex | texxet>
5104 \providecommand\bbl@provide@intraspace{}
5105 \bbl@trace{Redefinitions for bidi layout}
5106 \def\bbl@sspre@caption{% TODO: Unused!

```

```

5107 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
5108 \ifx\bbl@opt@layout@nnil\else % if layout=..
5109 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5110 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5111 \ifnum\bbl@bidimode>\z@ % TODO: always?
5112 \def\hangfrom#1{%
5113   \setbox\@tempboxa\hbox{#{#1}}%
5114   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5115   \noindent\box\@tempboxa}
5116 \def\raggedright{%
5117   \let\\\@centercr
5118   \bbl@startskip\z@skip
5119   \@rightskip\@flushglue
5120   \bbl@endskip\@rightskip
5121   \parindent\z@
5122   \parfillskip\bbl@startskip}
5123 \def\raggedleft{%
5124   \let\\\@centercr
5125   \bbl@startskip\@flushglue
5126   \bbl@endskip\z@skip
5127   \parindent\z@
5128   \parfillskip\bbl@endskip}
5129 \fi
5130 \IfBabelLayout{lists}
5131 {\bbl@sreplace\list
5132   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5133   \def\bbl@listleftmargin{%
5134     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5135   \ifcase\bbl@engine
5136     \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5137     \def\p@enumiii{\p@enumii}\theenumii{}\fi
5138   \fi
5139   \bbl@sreplace\@verbatim
5140     {\leftskip\@totalleftmargin}%
5141     {\bbl@startskip\textwidth
5142       \advance\bbl@startskip-\linewidth}%
5143   \bbl@sreplace\@verbatim
5144     {\rightskip\z@skip}%
5145     {\bbl@endskip\z@skip}}%
5146 {}
5147 \IfBabelLayout{contents}
5148 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5149   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5150 {}
5151 \IfBabelLayout{columns}
5152 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5153   \def\bbl@outputbox#1{%
5154     \hb@xt@\textwidth{%
5155       \hskip\columnwidth
5156       \hfil
5157       {\normalcolor\vrule \@width\columnseprule}%
5158       \hfil
5159       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5160       \hskip-\textwidth
5161       \hb@xt@\columnwidth{\box\@outputbox \hss}%
5162       \hskip\columnsep
5163       \hskip\columnwidth}}}%
5164 {}
5165 <<Footnote changes>>
5166 \IfBabelLayout{footnotes}%
5167 {\BabelFootnote\footnote\languagename{}\fi}%
5168 \BabelFootnote\localfootnote\languagename{}\fi}%
5169 \BabelFootnote\mainfootnote{}\fi}}

```

5170 {}

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.

```

5171 \IfBabelLayout{counters*}%
5172   {\bbl@add\bbl@opt@layout{.counters.}%
5173     \AddToHook{shipout/before}{%
5174       \let\bbl@tempa\babelsublr
5175       \let\babelsublr\@firstofone
5176       \let\bbl@save@thepage\thepage
5177       \protected@edef\thepage{\thepage}%
5178       \let\babelsublr\bbl@tempa}%
5179     \AddToHook{shipout/after}{%
5180       \let\thepage\bbl@save@thepage}}{}
5181 \IfBabelLayout{counters}%
5182   {\let\bbl@latinarabic=@arabic
5183     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5184     \let\bbl@asciroman=@roman
5185     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5186     \let\bbl@asciiRoman=@Roman
5187     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5188 \fi % end if layout
5189 </xetex | texpet>

```

## 10.2 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).

```

5190 <*texpet>
5191 \def\bbl@provide@extra#1{%
5192   % == auto-select encoding ==
5193   \ifx\bbl@encoding@select@off\empty\else
5194     \bbl@ifunset{\bbl@encoding@#1}%
5195     {\def\@elt##1{,##1,}%
5196       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5197       \count@\z@
5198       \bbl@foreach\bbl@tempe{%
5199         \def\bbl@tempd{##1}% Save last declared
5200         \advance\count@\@ne}%
5201       \ifnum\count@>\@ne % (1)
5202         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5203         \ifx\bbl@tempa\relax \let\bbl@tempa\empty \fi
5204         \bbl@replace\bbl@tempa{ },}%
5205         \global\bbl@csarg\let{encoding@#1}\empty
5206         \bbl@xin@{,\bbl@tempd,},{,\bbl@tempa,}%
5207         \ifin@else % if main encoding included in ini, do nothing
5208           \let\bbl@tempb\relax
5209           \bbl@foreach\bbl@tempa{%
5210             \ifx\bbl@tempb\relax
5211               \bbl@xin@{,##1,},{,\bbl@tempe,}%
5212               \ifin@\def\bbl@tempb{##1}\fi
5213             \fi}%
5214           \ifx\bbl@tempb\relax\else
5215             \bbl@exp{%
5216               \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>}%
5217             \gdef\<\bbl@encoding@#1>{%
5218               \\ \babel@save\\ \f@encoding
5219               \\ \bbl@add\\ \originalTeX\\ \selectfont}%
5220               \\ \fontencoding{\bbl@tempb}%
5221               \\ \selectfont}}%
5222           \fi
5223         \fi
5224       \fi}%

```

```

5225      {}%
5226    \fi}
5227 \</texxet>

```

### 10.3 LuaTeX

The loader for luatex is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the `hyphenmins` stuff, which is under the direct control of `babel`).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on `babel`, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by `babel`) provide a command to allocate them (although there are packages like `ctablestack`). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, `etex.sty` changes the way languages are allocated.

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the base option); (2) at `hyphen.cfg`, to modify some macros; (3) in the middle of `plain.def` and `babel.sty`, by `babel.def`, with the commands and other definitions for luatex (eg, `\babelpatterns`).

```

5228 \*luatex>
5229 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
5230 \bbl@trace{Read language.dat}
5231 \ifx\bbl@readstream\@undefined
5232   \csname newread\endcsname\bbl@readstream
5233 \fi
5234 \begingroup
5235   \toks@{}
5236   \count@ \z@ % 0=start, 1=0th, 2=normal
5237   \def\bbl@process@line#1#2 #3 #4 {%
5238     \ifx=#1%
5239       \bbl@process@synonym{#2}%
5240     \else
5241       \bbl@process@language{#1#2}{#3}{#4}%
5242     \fi
5243     \ignorespaces}
5244   \def\bbl@manylang{%
5245     \ifnum\bbl@last>\@ne
5246       \bbl@info{Non-standard hyphenation setup}%
5247     \fi
5248     \let\bbl@manylang\relax}
5249   \def\bbl@process@language#1#2#3{%
5250     \ifcase\count@
5251       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
5252     \or

```

```

5253 \count@ \tw@
5254 \fi
5255 \ifnum \count@=\tw@
5256 \expandafter \addlanguage \csname l@#1\endcsname
5257 \language \allocationnumber
5258 \chardef \bbl@last \allocationnumber
5259 \bbl@many lang
5260 \let \bbl@elt \relax
5261 \xdef \bbl@languages {%
5262 \bbl@languages \bbl@elt{#1}{\the \language}{#2}{#3}}%
5263 \fi
5264 \the \toks@
5265 \toks@{}
5266 \def \bbl@process@synonym@aux#1#2{%
5267 \global \expandafter \chardef \csname l@#1\endcsname#2 \relax
5268 \let \bbl@elt \relax
5269 \xdef \bbl@languages {%
5270 \bbl@languages \bbl@elt{#1}{#2}{}}}%
5271 \def \bbl@process@synonym#1{%
5272 \ifcase \count@
5273 \toks@ \expandafter {\the \toks@ \relax \bbl@process@synonym{#1}}%
5274 \or
5275 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5276 \else
5277 \bbl@process@synonym@aux{#1}{\the \bbl@last}%
5278 \fi}
5279 \ifx \bbl@languages \@undefined % Just a (sensible?) guess
5280 \chardef \l@english \z@
5281 \chardef \l@USenglish \z@
5282 \chardef \bbl@last \z@
5283 \global \@namedef{\bbl@hyphendata@0}{\hyphen.tex}{}
5284 \gdef \bbl@languages {%
5285 \bbl@elt{english}{0}{\hyphen.tex}{}%
5286 \bbl@elt{USenglish}{0}{}}%
5287 \else
5288 \global \let \bbl@languages @format \bbl@languages
5289 \def \bbl@elt#1#2#3#4{% Remove all except language 0
5290 \ifnum #2>\z@ \else
5291 \noexpand \bbl@elt{#1}{#2}{#3}{#4}%
5292 \fi}%
5293 \xdef \bbl@languages {\bbl@languages}%
5294 \fi
5295 \def \bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5296 \bbl@languages
5297 \openin \bbl@readstream=language.dat
5298 \ifeof \bbl@readstream
5299 \bbl@warning{I couldn't find language.dat. No additional\\%
5300 patterns loaded. Reported}%
5301 \else
5302 \loop
5303 \endlinechar \m@ne
5304 \read \bbl@readstream to \bbl@line
5305 \endlinechar ``^M
5306 \if T\ifeof \bbl@readstream F\fi T\relax
5307 \ifx \bbl@line \empty \else
5308 \edef \bbl@line {\bbl@line \space \space \space}%
5309 \expandafter \bbl@process@line \bbl@line \relax
5310 \fi
5311 \repeat
5312 \fi
5313 \closein \bbl@readstream
5314 \endgroup
5315 \bbl@trace{Macros for reading patterns files}

```



```

5316 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5317 \ifx\babelcatcodetablenum\undefined
5318   \ifx\newcatcodetable\undefined
5319     \def\babelcatcodetablenum{5211}
5320     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5321   \else
5322     \newcatcodetable\babelcatcodetablenum
5323     \newcatcodetable\bbl@pattcodes
5324   \fi
5325 \else
5326   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5327 \fi
5328 \def\bbl@luapatterns#1#2{%
5329   \bbl@get@enc#1::\@@
5330   \setbox\z@\hbox\bgroup
5331     \begingroup
5332       \savecatcodetable\babelcatcodetablenum\relax
5333       \initcatcodetable\bbl@pattcodes\relax
5334       \catcodetable\bbl@pattcodes\relax
5335       \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
5336       \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5337       \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5338       \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5339       \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5340       \catcode\`=12 \catcode\'=12 \catcode\"=12
5341       \input #1\relax
5342       \catcodetable\babelcatcodetablenum\relax
5343     \endgroup
5344     \def\bbl@tempa{#2}%
5345     \ifx\bbl@tempa\empty\else
5346       \input #2\relax
5347     \fi
5348   \egroup}%
5349 \def\bbl@patterns@lua#1{%
5350   \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5351     \csname l@#1\endcsname
5352     \edef\bbl@tempa{#1}%
5353   \else
5354     \csname l@#1:f@encoding\endcsname
5355     \edef\bbl@tempa{#1:f@encoding}%
5356   \fi\relax
5357   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5358   \@ifundefined{bbl@hyphendata@the\language}%
5359     {\def\bbl@elt##1##2##3##4{%
5360       \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5361       \def\bbl@tempb{##3}%
5362       \ifx\bbl@tempb\empty\else % if not a synonymous
5363         \def\bbl@tempc{{##3}{##4}}%
5364       \fi
5365       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5366     \fi}%
5367   \bbl@languages
5368   \@ifundefined{bbl@hyphendata@the\language}%
5369     {\bbl@info{No hyphenation patterns were set for\\%
5370       language '\bbl@tempa'. Reported}}%
5371     {\expandafter\expandafter\expandafter\bbl@luapatterns
5372       \csname bbl@hyphendata@the\language\endcsname}}}%
5373 \endinput\fi
5374 % Here ends \ifx\AddBabelHook\undefined
5375 % A few lines are only read by hyphen.cfg
5376 \ifx\DisableBabelHook\undefined
5377   \AddBabelHook{luatex}{everylanguage}{%
5378     \def\process@language##1##2##3{%

```

```

5379     \def\process@line####1####2 ####3 ####4 {}}}
5380 \AddBabelHook{luatex}{loadpatterns}{%
5381     \input #1\relax
5382     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5383         {{#1}}}}
5384 \AddBabelHook{luatex}{loadexceptions}{%
5385     \input #1\relax
5386     \def\bbl@tempb##1##2{{##1}{#1}}%
5387     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5388         {\expandafter\expandafter\expandafter\bbl@tempb
5389         \csname bbl@hyphendata@the\language\endcsname}}
5390 \endinput\fi
5391 % Here stops reading code for hyphen.cfg
5392 % The following is read the 2nd time it's loaded
5393 % First, global declarations for lua
5394 \begingroup % TODO - to a lua file
5395 \catcode`\%=12
5396 \catcode`\'=12
5397 \catcode`\ "=12
5398 \catcode`\:=12
5399 \directlua{
5400     Babel = Babel or {}
5401     function Babel.lua_error(e, a)
5402         tex.print([[ \noexpand\csname bbl@error\endcsname{]] ..
5403             e .. '}' .. (a or '') .. '}{}{}')
5404     end
5405     function Babel.bytes(line)
5406         return line:gsub(".",
5407             function (chr) return unicode.utf8.char(string.byte(chr)) end)
5408     end
5409     function Babel.begin_process_input()
5410         if luatexbase and luatexbase.add_to_callback then
5411             luatexbase.add_to_callback('process_input_buffer',
5412                 Babel.bytes, 'Babel.bytes')
5413         else
5414             Babel.callback = callback.find('process_input_buffer')
5415             callback.register('process_input_buffer', Babel.bytes)
5416         end
5417     end
5418     function Babel.end_process_input ()
5419         if luatexbase and luatexbase.remove_from_callback then
5420             luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5421         else
5422             callback.register('process_input_buffer', Babel.callback)
5423         end
5424     end
5425     function Babel.addpatterns(pp, lg)
5426         local lg = lang.new(lg)
5427         local pats = lang.patterns(lg) or ''
5428         lang.clear_patterns(lg)
5429         for p in pp:gmatch('[^%s]+') do
5430             ss = ''
5431             for i in string.utfcharacters(p:gsub('%d', '')) do
5432                 ss = ss .. '%d?' .. i
5433             end
5434             ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5435             ss = ss:gsub('%.%d%?$', '%%.')
5436             pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5437             if n == 0 then
5438                 tex.sprint(
5439                     [[ \string\csname\space bbl@info\endcsname{New pattern: ]]
5440                     .. p .. [[]]])
5441                 pats = pats .. ' ' .. p

```

```

5442     else
5443         tex.sprint(
5444             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
5445             .. p .. [{}]])
5446     end
5447 end
5448 lang.patterns(lg, pats)
5449 end
5450 Babel.characters = Babel.characters or {}
5451 Babel.ranges = Babel.ranges or {}
5452 function Babel.hlist_has_bidi(head)
5453     local has_bidi = false
5454     local ranges = Babel.ranges
5455     for item in node.traverse(head) do
5456         if item.id == node.id'glyph' then
5457             local itemchar = item.char
5458             local chardata = Babel.characters[itemchar]
5459             local dir = chardata and chardata.d or nil
5460             if not dir then
5461                 for nn, et in ipairs(ranges) do
5462                     if itemchar < et[1] then
5463                         break
5464                     elseif itemchar <= et[2] then
5465                         dir = et[3]
5466                         break
5467                     end
5468                 end
5469             end
5470             if dir and (dir == 'al' or dir == 'r') then
5471                 has_bidi = true
5472             end
5473         end
5474     end
5475     return has_bidi
5476 end
5477 function Babel.set_chranges_b (script, chrng)
5478     if chrng == '' then return end
5479     texio.write('Replacing ' .. script .. ' script ranges')
5480     Babel.script_blocks[script] = {}
5481     for s, e in string.gmatch(chrng..' ', '(.-%.-%.-%s') do
5482         table.insert(
5483             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5484     end
5485 end
5486 function Babel.discard_sublr(str)
5487     if str:find( [[\string\indexentry]] ) and
5488        str:find( [[\string\babelsublr]] ) then
5489         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5490             function(m) return m:sub(2,-2) end )
5491     end
5492     return str
5493 end
5494 }
5495 \endgroup
5496 \ifx\newattribute\undefined\else % Test for plain
5497     \newattribute\bbl@attr@locale
5498     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5499     \AddBabelHook{luatex}{beforeextras}{%
5500         \setattribute\bbl@attr@locale\localeid}
5501 \fi
5502 \def\BabelStringsDefault{unicode}
5503 \let\luabbl@stop\relax
5504 \AddBabelHook{luatex}{encodedcommands}{%

```

```

5505 \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5506 \ifx\bbl@tempa\bbl@tempb\else
5507   \directlua{Babel.begin_process_input()}%
5508   \def\luabbl@stop{%
5509     \directlua{Babel.end_process_input()}}%
5510 \fi}%
5511 \AddBabelHook{luatex}{stopcommands}{%
5512   \luabbl@stop
5513   \let\luabbl@stop\relax}
5514 \AddBabelHook{luatex}{patterns}{%
5515   \@ifundefined{bbl@hyphendata@the\language}%
5516     {\def\bbl@elt##1##2##3##4{%
5517       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5518       \def\bbl@tempb{##3}%
5519       \ifx\bbl@tempb@empty\else % if not a synonymous
5520         \def\bbl@tempc{{##3}{##4}}%
5521         \fi
5522         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5523       \fi}%
5524     \bbl@languages
5525     \@ifundefined{bbl@hyphendata@the\language}%
5526       {\bbl@info{No hyphenation patterns were set for\\
5527         language '#2'. Reported}}%
5528       {\expandafter\expandafter\expandafter\bbl@luapatterns
5529         \csname bbl@hyphendata@the\language\endcsname}}}%
5530   \@ifundefined{bbl@patterns@}{}%
5531   \begingroup
5532     \bbl@xin@{, \number\language, }{, \bbl@pttnlist}%
5533     \ifin@else
5534       \ifx\bbl@patterns@empty\else
5535         \directlua{ Babel.addpatterns(
5536           [[\bbl@patterns@]], \number\language) }%
5537       \fi
5538       \@ifundefined{bbl@patterns@#1}%
5539       \@empty
5540       {\directlua{ Babel.addpatterns(
5541         [[\space\csname bbl@patterns@#1\endcsname]],
5542         \number\language) }}%
5543       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5544     \fi
5545   \endgroup}%
5546 \bbl@exp{%
5547   \bbl@ifunset{bbl@prehc@the\languagename}{}%
5548   {\bbl@ifblank{\bbl@cs{prehc@the\languagename}}{}}%
5549   {\prehyphenchar=\bbl@c{prehc}\relax}}}%

```

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

```

5550 \@onlypreamble\babelpatterns
5551 \AtEndOfPackage{%
5552   \newcommand\babelpatterns[2][@empty]{%
5553     \ifx\bbl@patterns@relax
5554       \let\bbl@patterns@empty
5555     \fi
5556     \ifx\bbl@pttnlist@empty\else
5557       \bbl@warning{%
5558         You must not intermingle \string\selectlanguage\space and\\
5559         \string\babelpatterns\space or some patterns will not\\
5560         be taken into account. Reported}%
5561       \fi
5562     \ifx@empty#1%
5563       \protected@edef\bbl@patterns@{\bbl@patterns@space#2}%

```

```

5564 \else
5565 \edef\bbl@tempb{\zap@space#1 \@empty}%
5566 \bbl@for\bbl@tempa\bbl@tempb{%
5567 \bbl@fixname\bbl@tempa
5568 \bbl@iflanguage\bbl@tempa{%
5569 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5570 \ifundefined{bbl@patterns@\bbl@tempa}%
5571 \@empty
5572 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5573 #2}}}%
5574 \fi}}

```

## 10.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```

5575 % TODO - to a lua file
5576 \directlua{
5577   Babel = Babel or {}
5578   Babel.linebreaking = Babel.linebreaking or {}
5579   Babel.linebreaking.before = {}
5580   Babel.linebreaking.after = {}
5581   Babel.locale = {} % Free to use, indexed by \localeid
5582   function Babel.linebreaking.add_before(func, pos)
5583     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5584     if pos == nil then
5585       table.insert(Babel.linebreaking.before, func)
5586     else
5587       table.insert(Babel.linebreaking.before, pos, func)
5588     end
5589   end
5590   function Babel.linebreaking.add_after(func)
5591     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5592     table.insert(Babel.linebreaking.after, func)
5593   end
5594 }
5595 \def\bbl@intraspace#1 #2 #3\@@{%
5596 \directlua{
5597   Babel = Babel or {}
5598   Babel.intraspaces = Babel.intraspaces or {}
5599   Babel.intraspaces['\csname bbl@sbc@p\language\endcsname'] = %
5600     {b = #1, p = #2, m = #3}
5601   Babel.locale_props[\the\localeid].intraspace = %
5602     {b = #1, p = #2, m = #3}
5603 }}
5604 \def\bbl@intrapenalty#1\@@{%
5605 \directlua{
5606   Babel = Babel or {}
5607   Babel.intrapenalties = Babel.intrapenalties or {}
5608   Babel.intrapenalties['\csname bbl@sbc@p\language\endcsname'] = #1
5609   Babel.locale_props[\the\localeid].intrapenalty = #1
5610 }}
5611 \begingroup
5612 \catcode`\%=12
5613 \catcode`\&=14
5614 \catcode`\'=12
5615 \catcode`\~=12
5616 \gdef\bbl@seaintraspace{&
5617 \let\bbl@seaintraspace\relax
5618 \directlua{
5619   Babel = Babel or {}

```

```

5620 Babel.sea_enabled = true
5621 Babel.sea_ranges = Babel.sea_ranges or {}
5622 function Babel.set_chranges (script, chrng)
5623     local c = 0
5624     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5625         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5626         c = c + 1
5627     end
5628 end
5629 function Babel.sea_disc_to_space (head)
5630     local sea_ranges = Babel.sea_ranges
5631     local last_char = nil
5632     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5633     for item in node.traverse(head) do
5634         local i = item.id
5635         if i == node.id'glyph' then
5636             last_char = item
5637         elseif i == 7 and item.subtype == 3 and last_char
5638             and last_char.char > 0x0C99 then
5639             quad = font.getfont(last_char.font).size
5640             for lg, rg in pairs(sea_ranges) do
5641                 if last_char.char > rg[1] and last_char.char < rg[2] then
5642                     lg = lg:sub(1, 4) &% Remove trailing number of, eg, Cyril
5643                     local intraspace = Babel.intraspaces[lg]
5644                     local intrapenalty = Babel.intrapenalties[lg]
5645                     local n
5646                     if intrapenalty ~= 0 then
5647                         n = node.new(14, 0)      &% penalty
5648                         n.penalty = intrapenalty
5649                         node.insert_before(head, item, n)
5650                     end
5651                     n = node.new(12, 13)      &% (glue, spaceskip)
5652                     node.setglue(n, intraspace.b * quad,
5653                                 intraspace.p * quad,
5654                                 intraspace.m * quad)
5655                     node.insert_before(head, item, n)
5656                     node.remove(head, item)
5657                 end
5658             end
5659         end
5660     end
5661 end
5662 }&
5663 \bbl@luaohyphenate}

```

## 10.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

```

5664 \catcode`\%=14
5665 \gdef\bbl@cjkintraspaces{%
5666     \let\bbl@cjkintraspaces\relax
5667     \directlua{
5668         Babel = Babel or {}
5669         require('babel-data-cjk.lua')
5670         Babel.cjk_enabled = true
5671         function Babel.cjk_linebreak(head)
5672             local GLYPH = node.id'glyph'
5673             local last_char = nil

```

```

5674     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5675     local last_class = nil
5676     local last_lang = nil
5677
5678     for item in node.traverse(head) do
5679         if item.id == GLYPH then
5680
5681             local lang = item.lang
5682
5683             local LOCALE = node.get_attribute(item,
5684                 Babel.attr_locale)
5685             local props = Babel.locale_props[LOCALE]
5686
5687             local class = Babel.cjk_class[item.char].c
5688
5689             if props.cjk_quotes and props.cjk_quotes[item.char] then
5690                 class = props.cjk_quotes[item.char]
5691             end
5692
5693             if class == 'cp' then class = 'cl' end % ]] as CL
5694             if class == 'id' then class = 'I' end
5695
5696             local br = 0
5697             if class and last_class and Babel.cjk_breaks[last_class][class] then
5698                 br = Babel.cjk_breaks[last_class][class]
5699             end
5700
5701             if br == 1 and props.linebreak == 'c' and
5702                 lang ~= \the\l@nohyphenation\space and
5703                 last_lang ~= \the\l@nohyphenation then
5704                 local intrapenalty = props.intrapenalty
5705                 if intrapenalty ~= 0 then
5706                     local n = node.new(14, 0)      % penalty
5707                     n.penalty = intrapenalty
5708                     node.insert_before(head, item, n)
5709                 end
5710                 local intraspace = props.intraspace
5711                 local n = node.new(12, 13)      % (glue, spaceskip)
5712                 node.setglue(n, intraspace.b * quad,
5713                     intraspace.p * quad,
5714                     intraspace.m * quad)
5715                 node.insert_before(head, item, n)
5716             end
5717
5718             if font.getfont(item.font) then
5719                 quad = font.getfont(item.font).size
5720             end
5721             last_class = class
5722             last_lang = lang
5723             else % if penalty, glue or anything else
5724                 last_class = nil
5725             end
5726         end
5727         lang.hyphenate(head)
5728     end
5729 }%
5730 \bbl@luahyphenate}
5731 \gdef\bbl@luahyphenate{%
5732 \let\bbl@luahyphenate\relax
5733 \directlua{
5734     luatexbase.add_to_callback('hyphenate',
5735     function (head, tail)
5736         if Babel.linebreaking.before then

```

```

5737     for k, func in ipairs(Babel.linebreaking.before) do
5738         func(head)
5739     end
5740 end
5741 if Babel.cjk_enabled then
5742     Babel.cjk_linebreak(head)
5743 end
5744 lang.hyphenate(head)
5745 if Babel.linebreaking.after then
5746     for k, func in ipairs(Babel.linebreaking.after) do
5747         func(head)
5748     end
5749 end
5750 if Babel.sea_enabled then
5751     Babel.sea_disc_to_space(head)
5752 end
5753 end,
5754 'Babel.hyphenate')
5755 }
5756 }
5757 \endgroup
5758 \def\bbl@provide@intraspace{%
5759     \bbl@ifunset{\bbl@intsp@{language}}{%
5760         {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5761             \bbl@xin@{/c}{\bbl@cl{lnbrk}}}%
5762         \ifin@           % cjk
5763             \bbl@cjk@intraspace
5764             \directlua{
5765                 Babel = Babel or {}
5766                 Babel.locale_props = Babel.locale_props or {}
5767                 Babel.locale_props[\the\localeid].linebreak = 'c'
5768             }%
5769             \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5770             \ifx\bbl@KVP@intrapenalty\@nnil
5771                 \bbl@intrapenalty0\@{}
5772             \fi
5773         \else           % sea
5774             \bbl@seaintraspace
5775             \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5776             \directlua{
5777                 Babel = Babel or {}
5778                 Babel.sea_ranges = Babel.sea_ranges or {}
5779                 Babel.set_chranges('\bbl@cl{sbcp}',
5780                                     '\bbl@cl{chrng}')
5781             }%
5782             \ifx\bbl@KVP@intrapenalty\@nnil
5783                 \bbl@intrapenalty0\@{}
5784             \fi
5785         \fi
5786     \fi
5787     \ifx\bbl@KVP@intrapenalty\@nnil\else
5788         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@{}
5789     \fi}}

```

## 10.6 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-

```

5790 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5791 \def\bblar@chars{%
5792     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5793     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5794     0640,0641,0642,0643,0644,0645,0646,0647,0649}

```



```

5795 \def\bblar@elongated{%
5796   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5797   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5798   0649,064A}
5799 \begingroup
5800   \catcode\_ =11 \catcode\:=11
5801   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5802 \endgroup
5803 \gdef\bblar@arabicjust{% TODO. Allow for several locales.
5804   \let\bblar@arabicjust\relax
5805   \newattribute\bblar@kashida
5806   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5807   \bblar@kashida=\z@
5808   \bbl@patchfont{\bbl@parsejalt}}%
5809   \directlua{
5810     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5811     Babel.arabic.elong_map[\the\localeid] = {}
5812     luatexbase.add_to_callback('post_linebreak_filter',
5813       Babel.arabic.justify, 'Babel.arabic.justify')
5814     luatexbase.add_to_callback('hpack_filter',
5815       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5816   }}%

```

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

```

5817 \def\bblar@fetchjalt#1#2#3#4{%
5818   \bbl@exp{\bbl@foreach{#1}}{%
5819     \bbl@ifunset\bblar@JE@##1{%
5820       {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5821       {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5822     \directlua{%
5823       local last = nil
5824       for item in node.traverse(tex.box[0].head) do
5825         if item.id == node.id'glyph' and item.char > 0x600 and
5826           not (item.char == 0x200D) then
5827           last = item
5828         end
5829       end
5830       Babel.arabic.#3['##1#4'] = last.char
5831     }}

```

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

```

5832 \gdef\bblar@parsejalt{%
5833   \ifx\addfontfeature\@undefined\else
5834     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5835     \ifin@
5836       \directlua{%
5837         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5838           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5839           tex.print([\string\csname\space bbl@parsejalti\endcsname])
5840         end
5841       }%
5842     \fi
5843   \fi}
5844 \gdef\bblar@parsejalti{%
5845   \begingroup
5846     \let\bblar@parsejalt\relax % To avoid infinite loop
5847     \edef\bblar@tempb{\fontid\font}%
5848     \bblar@nofswarn
5849     \bblar@fetchjalt\bblar@elongated{}{from}}%
5850     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5851     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5852     \addfontfeature{RawFeature+=jalt}%
5853     % \@namedef\bblar@JE@0643{06AA}% todo: catch medial kaf

```

```

5854 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5855 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5856 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5857 \directlua{%
5858     for k, v in pairs(Babel.arabic.from) do
5859         if Babel.arabic.dest[k] and
5860             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5861             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5862                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5863         end
5864     end
5865 }%
5866 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5867 \begingroup
5868 \catcode`#=11
5869 \catcode`~ =11
5870 \directlua{
5871
5872 Babel.arabic = Babel.arabic or {}
5873 Babel.arabic.from = {}
5874 Babel.arabic.dest = {}
5875 Babel.arabic.justify_factor = 0.95
5876 Babel.arabic.justify_enabled = true
5877 Babel.arabic.kashida_limit = -1
5878
5879 function Babel.arabic.justify(head)
5880     if not Babel.arabic.justify_enabled then return head end
5881     for line in node.traverse_id(node.id'hlist', head) do
5882         Babel.arabic.justify_hlist(head, line)
5883     end
5884     return head
5885 end
5886
5887 function Babel.arabic.justify_hbox(head, gc, size, pack)
5888     local has_inf = false
5889     if Babel.arabic.justify_enabled and pack == 'exactly' then
5890         for n in node.traverse_id(l2, head) do
5891             if n.stretch_order > 0 then has_inf = true end
5892         end
5893         if not has_inf then
5894             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5895         end
5896     end
5897     return head
5898 end
5899
5900 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5901     local d, new
5902     local k_list, k_item, pos_inline
5903     local width, width_new, full, k_curr, wt_pos, goal, shift
5904     local subst_done = false
5905     local elong_map = Babel.arabic.elong_map
5906     local cnt
5907     local last_line
5908     local GLYPH = node.id'glyph'
5909     local KASHIDA = Babel.attr_kashida
5910     local LOCALE = Babel.attr_locale
5911
5912     if line == nil then
5913         line = {}
5914         line.glue_sign = 1

```

```

5915     line.glue_order = 0
5916     line.head = head
5917     line.shift = 0
5918     line.width = size
5919 end
5920
5921 % Exclude last line. todo. But-- it discards one-word lines, too!
5922 % ? Look for glue = 12:15
5923 if (line.glue_sign == 1 and line.glue_order == 0) then
5924     elongs = {}      % Stores elongated candidates of each line
5925     k_list = {}      % And all letters with kashida
5926     pos_inline = 0   % Not yet used
5927
5928     for n in node.traverse_id(GLYPH, line.head) do
5929         pos_inline = pos_inline + 1 % To find where it is. Not used.
5930
5931         % Elongated glyphs
5932         if elong_map then
5933             local locale = node.get_attribute(n, LOCALE)
5934             if elong_map[locale] and elong_map[locale][n.font] and
5935                 elong_map[locale][n.font][n.char] then
5936                 table.insert(elongs, {node = n, locale = locale} )
5937                 node.set_attribute(n.prev, KASHIDA, 0)
5938             end
5939         end
5940
5941         % Tatwil
5942         if Babel.kashida_wts then
5943             local k_wt = node.get_attribute(n, KASHIDA)
5944             if k_wt > 0 then % todo. parameter for multi inserts
5945                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5946             end
5947         end
5948
5949     end % of node.traverse_id
5950
5951     if #elongs == 0 and #k_list == 0 then goto next_line end
5952     full = line.width
5953     shift = line.shift
5954     goal = full * Babel.arabic.justify_factor % A bit crude
5955     width = node.dimensions(line.head) % The 'natural' width
5956
5957     % == Elongated ==
5958     % Original idea taken from 'chickenize'
5959     while (#elongs > 0 and width < goal) do
5960         subst_done = true
5961         local x = #elongs
5962         local curr = elongs[x].node
5963         local oldchar = curr.char
5964         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5965         width = node.dimensions(line.head) % Check if the line is too wide
5966         % Substitute back if the line would be too wide and break:
5967         if width > goal then
5968             curr.char = oldchar
5969             break
5970         end
5971         % If continue, pop the just substituted node from the list:
5972         table.remove(elongs, x)
5973     end
5974
5975     % == Tatwil ==
5976     if #k_list == 0 then goto next_line end
5977

```

```

5978 width = node.dimensions(line.head) % The 'natural' width
5979 k_curr = #k_list % Traverse backwards, from the end
5980 wt_pos = 1
5981
5982 while width < goal do
5983   subst_done = true
5984   k_item = k_list[k_curr].node
5985   if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5986     d = node.copy(k_item)
5987     d.char = 0x0640
5988     d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5989     d.xoffset = 0
5990     line.head, new = node.insert_after(line.head, k_item, d)
5991     width_new = node.dimensions(line.head)
5992     if width > goal or width == width_new then
5993       node.remove(line.head, new) % Better compute before
5994       break
5995     end
5996     if Babel.fix_diacr then
5997       Babel.fix_diacr(k_item.next)
5998     end
5999     width = width_new
6000   end
6001   if k_curr == 1 then
6002     k_curr = #k_list
6003     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6004   else
6005     k_curr = k_curr - 1
6006   end
6007 end
6008
6009 % Limit the number of tatweel by removing them. Not very efficient,
6010 % but it does the job in a quite predictable way.
6011 if Babel.arabic.kashida_limit > -1 then
6012   cnt = 0
6013   for n in node.traverse_id(GLYPH, line.head) do
6014     if n.char == 0x0640 then
6015       cnt = cnt + 1
6016       if cnt > Babel.arabic.kashida_limit then
6017         node.remove(line.head, n)
6018       end
6019     else
6020       cnt = 0
6021     end
6022   end
6023 end
6024
6025 ::next_line::
6026
6027 % Must take into account marks and ins, see luatex manual.
6028 % Have to be executed only if there are changes. Investigate
6029 % what's going on exactly.
6030 if subst_done and not gc then
6031   d = node.hpack(line.head, full, 'exactly')
6032   d.shift = shift
6033   node.insert_before(head, line, d)
6034   node.remove(head, line)
6035 end
6036 end % if process line
6037 end
6038 }
6039 \endgroup
6040 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

## 10.7 Common stuff

```
6041 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
6042 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
6043 \DisableBabelHook{babel-fontspec}
6044 <<Font selection>>
```

## 10.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a 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.

```
6045 % TODO - to a lua file
6046 \directlua{
6047 Babel.script_blocks = {
6048   ['dflt'] = {},
6049   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6050              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6051   ['Armn'] = {{0x0530, 0x058F}},
6052   ['Beng'] = {{0x0980, 0x09FF}},
6053   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6054   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6055   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6056              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6057   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6058   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6059              {0xAB00, 0xAB2F}},
6060   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6061   % Don't follow strictly Unicode, which places some Coptic letters in
6062   % the 'Greek and Coptic' block
6063   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6064   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6065              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6066              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6067              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6068              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6069              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6070   ['Hebr'] = {{0x0590, 0x05FF}},
6071   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6072              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6073   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6074   ['Knda'] = {{0x0C80, 0x0CFF}},
6075   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6076              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6077              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6078   ['Lao'] = {{0x0E80, 0x0EFF}},
6079   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6080              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6081              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6082   ['Mahj'] = {{0x11150, 0x1117F}},
6083   ['Mlym'] = {{0x0D00, 0x0D7F}},
6084   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6085   ['Orya'] = {{0x0B00, 0x0B7F}},
6086   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6087   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6088   ['Taml'] = {{0x0B80, 0x0BFF}},
6089   ['Telu'] = {{0x0C00, 0x0C7F}},
6090   ['Tfng'] = {{0x2D30, 0x2D7F}},
6091   ['Thai'] = {{0x0E00, 0x0E7F}},
```

```

6092 ['Tibt'] = {{0x0F00, 0x0FFF}},
6093 ['Vaii'] = {{0xA500, 0xA63F}},
6094 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6095 }
6096
6097 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6098 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6099 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6100
6101 function Babel.locale_map(head)
6102   if not Babel.locale_mapped then return head end
6103
6104   local LOCALE = Babel.attr_locale
6105   local GLYPH = node.id('glyph')
6106   local inmath = false
6107   local toloc_save
6108   for item in node.traverse(head) do
6109     local toloc
6110     if not inmath and item.id == GLYPH then
6111       % Optimization: build a table with the chars found
6112       if Babel.chr_to_loc[item.char] then
6113         toloc = Babel.chr_to_loc[item.char]
6114       else
6115         for lc, maps in pairs(Babel.loc_to_scr) do
6116           for _, rg in pairs(maps) do
6117             if item.char >= rg[1] and item.char <= rg[2] then
6118               Babel.chr_to_loc[item.char] = lc
6119               toloc = lc
6120               break
6121             end
6122           end
6123         end
6124         % Treat composite chars in a different fashion, because they
6125         % 'inherit' the previous locale.
6126         if (item.char >= 0x0300 and item.char <= 0x036F) or
6127            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6128            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6129           Babel.chr_to_loc[item.char] = -2000
6130           toloc = -2000
6131         end
6132         if not toloc then
6133           Babel.chr_to_loc[item.char] = -1000
6134         end
6135       end
6136       if toloc == -2000 then
6137         toloc = toloc_save
6138       elseif toloc == -1000 then
6139         toloc = nil
6140       end
6141       if toloc and Babel.locale_props[toloc] and
6142          Babel.locale_props[toloc].letters and
6143          tex.getcatcode(item.char) \string~= 11 then
6144         toloc = nil
6145       end
6146       if toloc and Babel.locale_props[toloc].script
6147          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6148          and Babel.locale_props[toloc].script ==
6149          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6150         toloc = nil
6151       end
6152       if toloc then
6153         if Babel.locale_props[toloc].lg then
6154           item.lang = Babel.locale_props[toloc].lg

```

```

6155         node.set_attribute(item, LOCALE, toloc)
6156     end
6157     if Babel.locale_props[toloc]['/'..item.font] then
6158         item.font = Babel.locale_props[toloc]['/'..item.font]
6159     end
6160 end
6161 toloc_save = toloc
6162 elseif not inmath and item.id == 7 then % Apply recursively
6163     item.replace = item.replace and Babel.locale_map(item.replace)
6164     item.pre      = item.pre and Babel.locale_map(item.pre)
6165     item.post     = item.post and Babel.locale_map(item.post)
6166 elseif item.id == node.id'math' then
6167     inmath = (item.subtype == 0)
6168 end
6169 end
6170 return head
6171 end
6172 }

```

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

```

6173 \newcommand\babelcharproperty[1]{%
6174   \count@=#1\relax
6175   \ifvmode
6176     \expandafter\bbl@chprop
6177   \else
6178     \bbl@error{charproperty-only-vertical}{}{}{}%
6179   \fi}
6180 \newcommand\bbl@chprop[3][\the\count@]{%
6181   \@tempcnta=#1\relax
6182   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6183   {\bbl@error{unknown-char-property}{}{#2}{}%
6184   }%
6185   \loop
6186     \bbl@cs{chprop@#2}{#3}%
6187     \ifnum\count@<\@tempcnta
6188       \advance\count@\@ne
6189     \repeat}
6190 \def\bbl@chprop@direction#1{%
6191   \directlua{
6192     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6193     Babel.characters[\the\count@]['d'] = '#1'
6194   }}
6195 \let\bbl@chprop@bc\bbl@chprop@direction
6196 \def\bbl@chprop@mirror#1{%
6197   \directlua{
6198     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6199     Babel.characters[\the\count@]['m'] = '\number#1'
6200   }}
6201 \let\bbl@chprop@bmg\bbl@chprop@mirror
6202 \def\bbl@chprop@linebreak#1{%
6203   \directlua{
6204     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6205     Babel.cjk_characters[\the\count@]['c'] = '#1'
6206   }}
6207 \let\bbl@chprop@lb\bbl@chprop@linebreak
6208 \def\bbl@chprop@locale#1{%
6209   \directlua{
6210     Babel.chr_to_loc = Babel.chr_to_loc or {}
6211     Babel.chr_to_loc[\the\count@] =
6212       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6213   }}

```

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.

```
6214 \directlua{
6215   Babel.nohyphenation = \the\l@nohyphenation
6216 }
```

Now the  $\TeX$  high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the  $\{n\}$  syntax. For example,  $\text{pre}=\{1\}\{1\}$  becomes  $\text{function}(m) \text{ return } m[1]..m[1]..'-' \text{ end}$ , where  $m$  are the matches returned after applying the pattern. With a mapped capture the functions are similar to  $\text{function}(m) \text{ return } \text{Babel.capt\_map}(m[1],1) \text{ end}$ , where the last argument identifies the mapping to be applied to  $m[1]$ . The way it is carried out is somewhat tricky, but the effect is not dissimilar to  $\text{lua load}$  – save the code as string in a  $\TeX$  macro, and expand this macro at the appropriate place. As  $\text{\directlua}$  does not take into account the current catcode of  $\text{\@}$ , we just avoid this character in macro names (which explains the internal group, too).

```
6217 \begingroup
6218 \catcode`\~ = 12
6219 \catcode`\% = 12
6220 \catcode`\& = 14
6221 \catcode`\| = 12
6222 \gdef\babelprehyphenation{%&
6223   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{]}
6224 \gdef\babelposthyphenation{%&
6225   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}{]}
6226 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6227   \ifcase#1
6228     \bbl@activateprehyphen
6229   \or
6230     \bbl@activateposthyphen
6231   \fi
6232 \begingroup
6233   \def\babeltempa{\bbl@add@list\babeltempb}%&
6234   \let\babeltempb@empty
6235   \def\bbl@tempa{#5}%&
6236   \bbl@replace\bbl@tempa{,}{ ,}%& TODO. Ugly trick to preserve {}
6237   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6238     \bbl@ifsamestring{##1}{remove}%&
6239     {\bbl@add@list\babeltempb{nil}}}%&
6240   {\directlua{
6241     local rep = {[##1]=}
6242     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6243     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6244     rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6245     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6246     rep = rep:gsub('node%s*=%s*([^\s,]*)', Babel.capture_node)
6247     rep = rep:gsub(&
6248       '(norule)%s*=%s*([^\s,]*)%s+([^\s,]*)%s+([^\s,]*)',
6249       'norule = { ' .. '%2, %3, %4' .. ' }')
6250     if #1 == 0 or #1 == 2 then
6251       rep = rep:gsub(&
6252         '(space)%s*=%s*([^\s,]*)%s+([^\s,]*)%s+([^\s,]*)',
6253         'space = { ' .. '%2, %3, %4' .. ' }')
6254       rep = rep:gsub(&
6255         '(spacefactor)%s*=%s*([^\s,]*)%s+([^\s,]*)%s+([^\s,]*)',
6256         'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6257       rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6258     else
6259       rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6260       rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6261       rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6262     end
6263     tex.print([[string\babeltempa{[]] .. rep .. [{}]])
6264   }}%&
6265   \bbl@foreach\babeltempb{%&
```



```

6266 \bbl@forkv{##1}}{&%
6267 \in{,###1,},{,nil,step,data,remove,insert,string,no,pre,no,&%
6268 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6269 \ifin@else
6270 \bbl@error{bad-transform-option}{###1}{}}{&%
6271 \fi}}&%
6272 \let\bbl@kv@attribute\relax
6273 \let\bbl@kv@label\relax
6274 \let\bbl@kv@fonts\@empty
6275 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6276 \ifx\bbl@kv@fonts\@empty\else\bbl@settransform\fi
6277 \ifx\bbl@kv@attribute\relax
6278 \ifx\bbl@kv@label\relax\else
6279 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6280 \bbl@replace\bbl@kv@fonts{ }{,}&%
6281 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6282 \count@z@
6283 \def\bbl@elt##1##2##3{&%
6284 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6285 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6286 {\count@z@ne}&%
6287 {\bbl@error{font-conflict-transforms}{}}{}}}&%
6288 {}}&%
6289 \bbl@transform@list
6290 \ifnum\count@z@
6291 \bbl@exp{\global\bbl@add\bbl@transform@list
6292 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6293 \fi
6294 \bbl@ifunset{\bbl@kv@attribute}&%
6295 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6296 {}&%
6297 \global\bbl@carg\setattribute{\bbl@kv@attribute}@ne
6298 \fi
6299 \else
6300 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6301 \fi
6302 \directlua{
6303 local lbkr = Babel.linebreaking.replacements[#1]
6304 local u = unicode.utf8
6305 local id, attr, label
6306 if #1 == 0 then
6307 id = \the\csname bbl@id@#3\endcsname\space
6308 else
6309 id = \the\csname l@#3\endcsname\space
6310 end
6311 \ifx\bbl@kv@attribute\relax
6312 attr = -1
6313 \else
6314 attr = luatexbase.registernumber'\bbl@kv@attribute'
6315 \fi
6316 \ifx\bbl@kv@label\relax\else &% Same refs:
6317 label = [==[\bbl@kv@label]==]
6318 \fi
6319 &% Convert pattern:
6320 local patt = string.gsub([==[#4]==], '%s', '')
6321 if #1 == 0 then
6322 patt = string.gsub(patt, '|', ' ')
6323 end
6324 if not u.find(patt, '()', nil, true) then
6325 patt = '()' .. patt .. '()'
6326 end
6327 if #1 == 1 then
6328 patt = string.gsub(patt, '%(%)%^', '^()')

```

```

6329     patt = string.gsub(patt, '%$%(%)', '()$')
6330 end
6331 patt = u.gsub(patt, '{(.)}',
6332     function (n)
6333         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6334     end)
6335 patt = u.gsub(patt, '{(%X%X%X%X+)}',
6336     function (n)
6337         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6338     end)
6339 lbkr[id] = lbkr[id] or {}
6340 table.insert(lbkr[id],
6341     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6342 }&%
6343 \endgroup}
6344 \endgroup
6345 \let\bbl@transfont@list@empty
6346 \def\bbl@settransfont{%
6347     \global\let\bbl@settransfont\relax % Execute only once
6348     \gdef\bbl@transfont{%
6349         \def\bbl@elt####1####2####3{%
6350             \bbl@ifblank{####3}%
6351             {\count@tw@}% Do nothing if no fonts
6352             {\count@z@
6353             \bbl@vforeach{####3}{%
6354                 \def\bbl@tempd{#####1}%
6355                 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6356                 \ifx\bbl@tempd\bbl@tempe
6357                     \count@one
6358                 \else\ifx\bbl@tempd\bbl@transfam
6359                     \count@one
6360                 \fi\fi}%
6361                 \ifcase\count@
6362                     \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6363                 \or
6364                     \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6365                 \fi}}%
6366             \bbl@transfont@list}%
6367     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6368     \gdef\bbl@transfam{-unknown-}%
6369     \bbl@foreach\bbl@font@fams{%
6370         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6371         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6372         {\xdef\bbl@transfam{##1}}%
6373     }}}}
6374 \DeclareRobustCommand\enablelocaletransform[1]{%
6375     \bbl@ifunset{\bbl@ATR@#1@language @}%
6376     {\bbl@error{transform-not-available}{#1}{}}}%
6377     {\bbl@csarg\setattribute{ATR@#1@language @}\@ne}}
6378 \DeclareRobustCommand\disablelocaletransform[1]{%
6379     \bbl@ifunset{\bbl@ATR@#1@language @}%
6380     {\bbl@error{transform-not-available-b}{#1}{}}}%
6381     {\bbl@csarg\unsetattribute{ATR@#1@language @}}}%
6382 \def\bbl@activateposthyphen{%
6383     \let\bbl@activateposthyphen\relax
6384     \directlua{
6385         require('babel-transforms.lua')
6386         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6387     }}
6388 \def\bbl@activateprehyphen{%
6389     \let\bbl@activateprehyphen\relax
6390     \directlua{
6391         require('babel-transforms.lua')

```

```

6392 Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6393 }}

```

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.

```

6394 \newcommand\localeprehyphenation[1]{%
6395 \directlua{ Babel.string_prehyphenation([=[#1]=], \the\localeid) }}

```

## 10.9 Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before luaotfload is applied, which is loaded by default by L<sup>A</sup>T<sub>E</sub>X. Just in case, consider the possibility it has not been loaded.

```

6396 \def\bbl@activate@preotf{%
6397 \let\bbl@activate@preotf\relax % only once
6398 \directlua{
6399 Babel = Babel or {}
6400 %
6401 function Babel.pre_otfload_v(head)
6402   if Babel.numbers and Babel.digits_mapped then
6403     head = Babel.numbers(head)
6404   end
6405   if Babel.bidi_enabled then
6406     head = Babel.bidi(head, false, dir)
6407   end
6408   return head
6409 end
6410 %
6411 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6412   if Babel.numbers and Babel.digits_mapped then
6413     head = Babel.numbers(head)
6414   end
6415   if Babel.bidi_enabled then
6416     head = Babel.bidi(head, false, dir)
6417   end
6418   return head
6419 end
6420 %
6421 luatexbase.add_to_callback('pre_linebreak_filter',
6422   Babel.pre_otfload_v,
6423   'Babel.pre_otfload_v',
6424   luatexbase.priority_in_callback('pre_linebreak_filter',
6425     'luaotfload.node_processor') or nil)
6426 %
6427 luatexbase.add_to_callback('hpack_filter',
6428   Babel.pre_otfload_h,
6429   'Babel.pre_otfload_h',
6430   luatexbase.priority_in_callback('hpack_filter',
6431     'luaotfload.node_processor') or nil)
6432 }}

```

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, but it's kept in basic-r.

```

6433 \breakafterdirmode=1
6434 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6435 \let\bbl@beforeforeign\leavevmode
6436 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6437 \RequirePackage{luatexbase}

```

```

6438 \bbl@activate@preotf
6439 \directlua{
6440   require('babel-data-bidi.lua')
6441   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6442   require('babel-bidi-basic.lua')
6443   \or
6444   require('babel-bidi-basic-r.lua')
6445   table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6446   table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6447   table.insert(Babel.ranges, {0x10000, 0x10FFFD, 'on'})
6448 \fi}
6449 \newattribute\bbl@attr@dir
6450 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6451 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6452 \fi
6453 \chardef\bbl@thetextdir\z@
6454 \chardef\bbl@thepardir\z@
6455 \def\bbl@getluadir#1{%
6456   \directlua{
6457     if tex.#1dir == 'TLT' then
6458       tex.sprint('0')
6459     elseif tex.#1dir == 'TRT' then
6460       tex.sprint('1')
6461     end}}
6462 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6463   \ifcase#3\relax
6464     \ifcase\bbl@getluadir{#1}\relax\else
6465       #2 TLT\relax
6466     \fi
6467   \else
6468     \ifcase\bbl@getluadir{#1}\relax
6469       #2 TRT\relax
6470     \fi
6471   \fi}
6472 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6473 \def\bbl@thedir{0}
6474 \def\bbl@textdir#1{%
6475   \bbl@setluadir{text}\textdir{#1}%
6476   \chardef\bbl@thetextdir#1\relax
6477   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6478   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6479 \def\bbl@pardir#1{% Used twice
6480   \bbl@setluadir{par}\pardir{#1}%
6481   \chardef\bbl@thepardir#1\relax}
6482 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6483 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6484 \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.

```

6485 \ifnum\bbl@bidimode>\z@ % Any bidi=
6486 \def\bbl@insidemath{0}%
6487 \def\bbl@everymath{\def\bbl@insidemath{1}}
6488 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6489 \frozen@everymath\expandafter{%
6490   \expandafter\bbl@everymath\the\frozen@everymath}
6491 \frozen@everydisplay\expandafter{%
6492   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6493 \AtBeginDocument{
6494   \directlua{
6495     function Babel.math_box_dir(head)
6496       if not (token.get_macro('bbl@insidemath') == '0') then
6497         if Babel.hlist_has_bidi(head) then

```

```

6498         local d = node.new(node.id'dir')
6499         d.dir = '+TRT'
6500         node.insert_before(head, node.has_glyph(head), d)
6501         local inmath = false
6502         for item in node.traverse(head) do
6503             if item.id == 11 then
6504                 inmath = (item.subtype == 0)
6505             elseif not inmath then
6506                 node.set_attribute(item,
6507                     Babel.attr_dir, token.get_macro('bbl@thedir'))
6508             end
6509         end
6510     end
6511 end
6512 return head
6513 end
6514 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6515     "Babel.math_box_dir", 0)
6516 if Babel.unset_atdir then
6517     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6518         "Babel.unset_atdir")
6519     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6520         "Babel.unset_atdir")
6521 end
6522 }%
6523 \fi

```

Experimental. Tentative name.

```

6524 \DeclareRobustCommand\localebox[1]{%
6525     {\def\bbl@insidemath{0}%
6526         \mbox{\foreignlanguage{\language}{#1}}}}

```

## 10.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

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.

```

6527 \bbl@trace{Redefinitions for bidi layout}
6528 %
6529 <<(*More package options)>> ≡
6530 \chardef\bbl@eqnpos\z@
6531 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6532 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6533 <</More package options>>
6534 %
6535 \ifnum\bbl@bidimode>\z@ % Any bidi=
6536     \matheqdirmode\@ne % A luatex primitive

```

```

6537 \let\bbl@eqnodir\relax
6538 \def\bbl@eqdel{()}
6539 \def\bbl@eqnum{%
6540   {\normalfont\normalcolor
6541     \expandafter\@firstoftwo\bbl@eqdel
6542     \theequation
6543     \expandafter\@secondoftwo\bbl@eqdel}}
6544 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6545 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6546 \def\bbl@eqno@flip#1{%
6547   \ifdim\predisplaysize=-\maxdimen
6548     \eqno
6549     \hb@xt@.01pt{%
6550       \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6551   \else
6552     \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6553   \fi
6554   \bbl@exp{\def\\@currentlabel{[\bbl@upset]}}
6555 \def\bbl@leqno@flip#1{%
6556   \ifdim\predisplaysize=-\maxdimen
6557     \leqno
6558     \hb@xt@.01pt{%
6559       \hss\hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}\hss}}%
6560   \else
6561     \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6562   \fi
6563   \bbl@exp{\def\\@currentlabel{[\bbl@upset]}}
6564 \AtBeginDocument{%
6565   \ifx\bbl@noamsmath\relax\else
6566   \ifx\maketag@@@\undefined % Normal equation, eqnarray
6567     \AddToHook{env/equation/begin}{%
6568       \ifnum\bbl@thetextdir>\z@
6569         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6570         \let\@eqnnum\bbl@eqnum
6571         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6572         \chardef\bbl@thetextdir\z@
6573         \bbl@add\normalfont{\bbl@eqnodir}%
6574         \ifcase\bbl@eqnpos
6575           \let\bbl@puteqno\bbl@eqno@flip
6576         \or
6577           \let\bbl@puteqno\bbl@leqno@flip
6578         \fi
6579       \fi}%
6580   \ifnum\bbl@eqnpos=\tw@ \else
6581     \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6582   \fi
6583   \AddToHook{env/eqnarray/begin}{%
6584     \ifnum\bbl@thetextdir>\z@
6585       \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6586       \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6587       \chardef\bbl@thetextdir\z@
6588       \bbl@add\normalfont{\bbl@eqnodir}%
6589       \ifnum\bbl@eqnpos=\@ne
6590         \def\@eqnnum{%
6591           \setbox\z@\hbox{\bbl@eqnum}%
6592           \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6593       \else
6594         \let\@eqnnum\bbl@eqnum
6595       \fi
6596     \fi}
6597   % Hack. YA luatex bug?:
6598   \expandafter\bbl@replace\csname] \endcsname{${}\eqno\kern.001pt${}%
6599 \else % amstex

```

```

6600 \bbl@exp{% Hack to hide maybe undefined conditionals:
6601 \chardef\bbl@eqnpos=0%
6602 \<iftagsleft>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6603 \ifnum\bbl@eqnpos=\@ne
6604 \let\bbl@ams@lap\hbox
6605 \else
6606 \let\bbl@ams@lap\llap
6607 \fi
6608 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6609 \bbl@sreplace\intertext@\{normalbaselines}%
6610 \{normalbaselines
6611 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6612 \ExplSyntaxOff
6613 \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|lap|flip
6614 \ifx\bbl@ams@lap\hbox % leqno
6615 \def\bbl@ams@flip#1{%
6616 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6617 \else % eqno
6618 \def\bbl@ams@flip#1{%
6619 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6620 \fi
6621 \def\bbl@ams@preset#1{%
6622 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6623 \ifnum\bbl@thetextdir>\z@
6624 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6625 \bbl@sreplace\textdef@\{hbox}\bbl@ams@tagbox\hbox}%
6626 \bbl@sreplace\maketag@@@\{hbox}\bbl@ams@tagbox#1}%
6627 \fi}%
6628 \ifnum\bbl@eqnpos=\tw@ \else
6629 \def\bbl@ams@equation{%
6630 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6631 \ifnum\bbl@thetextdir>\z@
6632 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6633 \chardef\bbl@thetextdir\z@
6634 \bbl@add\normalfont{\bbl@eqnodir}%
6635 \ifcase\bbl@eqnpos
6636 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6637 \or
6638 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6639 \fi
6640 \fi}%
6641 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6642 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6643 \fi
6644 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6645 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6646 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6647 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6648 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6649 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6650 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6651 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6652 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6653 % Hackish, for proper alignment. Don't ask me why it works!:
6654 \bbl@exp{% Avoid a 'visible' conditional
6655 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6656 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6657 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6658 \AddToHook{env/split/before}{%
6659 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6660 \ifnum\bbl@thetextdir>\z@
6661 \bbl@ifsamestring\@currentenv{equation}%
6662 {\ifx\bbl@ams@lap\hbox % leqno

```

```

6663         \def\bb@ams@flip#1{%
6664             \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6665         \else
6666             \def\bb@ams@flip#1{%
6667                 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6668             \fi}%
6669         {}%
6670     \fi}%
6671 \fi\fi}
6672 \fi
6673 \def\bb@provide@extra#1{%
6674 % == Counters: mapdigits ==
6675 % Native digits
6676 \ifx\bb@KVP@mapdigits\@nnil\else
6677     \bb@ifunset{\bb@dgnat@\language\name}{}%
6678     {\RequirePackage{luatexbase}%
6679      \bb@activate@preotf
6680      \directlua{
6681          Babel = Babel or {} %%% -> presets in luababel
6682          Babel.digits_mapped = true
6683          Babel.digits = Babel.digits or {}
6684          Babel.digits[\the\localeid] =
6685              table.pack(string.utfvalue('\bb@cl{dgnat}'))
6686          if not Babel.numbers then
6687              function Babel.numbers(head)
6688                  local LOCALE = Babel.attr_locale
6689                  local GLYPH = node.id'glyph'
6690                  local inmath = false
6691                  for item in node.traverse(head) do
6692                      if not inmath and item.id == GLYPH then
6693                          local temp = node.get_attribute(item, LOCALE)
6694                          if Babel.digits[temp] then
6695                              local chr = item.char
6696                              if chr > 47 and chr < 58 then
6697                                  item.char = Babel.digits[temp][chr-47]
6698                              end
6699                          end
6700                      elseif item.id == node.id'math' then
6701                          inmath = (item.subtype == 0)
6702                      end
6703                  end
6704                  return head
6705              end
6706          end
6707      }}%
6708 \fi
6709 % == transforms ==
6710 \ifx\bb@KVP@transforms\@nnil\else
6711     \def\bb@elt##1##2##3{%
6712         \in@{$transforms.}{{##1}}%
6713         \ifin@
6714             \def\bb@tempa{##1}%
6715             \bb@replace\bb@tempa{transforms.}{}%
6716             \bb@carg\bb@transforms{babel\bb@tempa}{{##2}{{##3}}}%
6717         \fi}%
6718     \csname bbl@inidata@\language\name\endcsname
6719     \bb@release@transforms\relax % \relax closes the last item.
6720 \fi}
6721 % Start tabular here:
6722 \def\localerestoredirs{%
6723     \ifcase\bb@thetextdir
6724     \ifnum\textdirection=\z@\else\textdir TLT\fi
6725     \else

```



```

6726 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6727 \fi
6728 \ifcase\bbbl@thepardir
6729 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6730 \else
6731 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6732 \fi}
6733 \IfBabelLayout{tabular}%
6734 {\chardef\bbbl@tabular@mode\tw@}% All RTL
6735 {\IfBabelLayout{notabular}%
6736 {\chardef\bbbl@tabular@mode\z@}%
6737 {\chardef\bbbl@tabular@mode\@ne}}% Mixed, with LTR cols
6738 \ifnum\bbbl@bidimode>\@ne % Any lua bidi= except default=1
6739 \ifcase\bbbl@tabular@mode\or % 1
6740 \let\bbbl@parabefore\relax
6741 \AddToHook{para/before}{\bbbl@parabefore}
6742 \AtBeginDocument{%
6743 \bbbl@replace\@tabular{$}{$}%
6744 \def\bbbl@insidemath{0}%
6745 \def\bbbl@parabefore{\localerestoredirs}}%
6746 \ifnum\bbbl@tabular@mode=\@ne
6747 \bbbl@ifunset{@tabclassz}{}%
6748 \bbbl@exp{% Hide conditionals
6749 \\\bbbl@sreplace\\ \@tabclassz
6750 {\<ifcase>\\ \@chnum}%
6751 {\\\localerestoredirs\<ifcase>\\ \@chnum}}}%
6752 \@ifpackageloaded{colortbl}%
6753 {\bbbl@sreplace\@classz
6754 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6755 {\@ifpackageloaded{array}%
6756 {\bbbl@exp{% Hide conditionals
6757 \\\bbbl@sreplace\\ \@classz
6758 {\<ifcase>\\ \@chnum}%
6759 {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6760 \\\bbbl@sreplace\\ \@classz
6761 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6762 {}}}%
6763 \fi}%
6764 \or % 2
6765 \let\bbbl@parabefore\relax
6766 \AddToHook{para/before}{\bbbl@parabefore}%
6767 \AtBeginDocument{%
6768 \@ifpackageloaded{colortbl}%
6769 {\bbbl@replace\@tabular{$}{$}%
6770 \def\bbbl@insidemath{0}%
6771 \def\bbbl@parabefore{\localerestoredirs}}%
6772 \bbbl@sreplace\@classz
6773 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6774 {}}}%
6775 \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.

```

6776 \AtBeginDocument{%
6777 \ifpackageloaded{multicol}%
6778 {\toks@ \expandafter{\multi@column@out}%
6779 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6780 {}}%
6781 \ifpackageloaded{paracol}%
6782 {\edef\pcol@output{%
6783 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6784 {}}%

```

```

6785 \fi
6786 \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`.

```

6787 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6788 \def\bbbl@nextfake#1{% non-local changes, use always inside a group!
6789 \bbbl@exp{%
6790 \def\bbbl@insidemath{0}%
6791 \mathdir\the\bodydir
6792 #1% Once entered in math, set boxes to restore values
6793 \<ifmmode>%
6794 \everyvbox{%
6795 \the\everyvbox
6796 \bodydir\the\bodydir
6797 \mathdir\the\mathdir
6798 \everyhbox{\the\everyhbox}%
6799 \everyvbox{\the\everyvbox}}%
6800 \everyhbox{%
6801 \the\everyhbox
6802 \bodydir\the\bodydir
6803 \mathdir\the\mathdir
6804 \everyhbox{\the\everyhbox}%
6805 \everyvbox{\the\everyvbox}}%
6806 \<fi>}}%
6807 \def\@hangfrom#1{%
6808 \setbox\@tempboxa\hbox{#1}%
6809 \hangindent\wd\@tempboxa
6810 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
6811 \shapemode\@ne
6812 \fi
6813 \noindent\box\@tempboxa}
6814 \fi
6815 \IfBabelLayout{tabular}
6816 {\let\bbbl@OL@tabular\@tabular
6817 \bbbl@replace\@tabular{$}{\bbbl@nextfake$}%
6818 \let\bbbl@NL@tabular\@tabular
6819 \AtBeginDocument{%
6820 \ifx\bbbl@NL@tabular\@tabular\else
6821 \bbbl@exp{\in{\bbbl@nextfake}{\@tabular}}}%
6822 \ifin\else
6823 \bbbl@replace\@tabular{$}{\bbbl@nextfake$}%
6824 \fi
6825 \let\bbbl@NL@tabular\@tabular
6826 \fi}}
6827 {}
6828 \IfBabelLayout{lists}
6829 {\let\bbbl@OL@list\list
6830 \bbbl@sreplace\list{\parshape}{\bbbl@listparshape}%
6831 \let\bbbl@NL@list\list
6832 \def\bbbl@listparshape#1#2#3{%
6833 \parshape #1 #2 #3 %
6834 \ifnum\bbbl@getluadir{page}=\bbbl@getluadir{par}\else
6835 \shapemode\tw@
6836 \fi}}
6837 {}
6838 \IfBabelLayout{graphics}
6839 {\let\bbbl@pictresetdir\relax
6840 \def\bbbl@pictsetdir#1{%
6841 \ifcase\bbbl@thetextdir
6842 \let\bbbl@pictresetdir\relax

```

```

6843 \else
6844 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6845 \or\textdir TLT
6846 \else\bodydir TLT \textdir TLT
6847 \fi
6848 % \(\text|par)dir required in pgf:
6849 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6850 \fi}%
6851 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw}%
6852 \directlua{
6853 Babel.get_picture_dir = true
6854 Babel.picture_has_bidi = 0
6855 %
6856 function Babel.picture_dir (head)
6857 if not Babel.get_picture_dir then return head end
6858 if Babel.hlist_has_bidi(head) then
6859 Babel.picture_has_bidi = 1
6860 end
6861 return head
6862 end
6863 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6864 "Babel.picture_dir")
6865 }%
6866 \AtBeginDocument{%
6867 \def\LS@rot{%
6868 \setbox\@outputbox\vbox{%
6869 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6870 \long\def\put(#1,#2)#3{%
6871 \@killglue
6872 % Try:
6873 \ifx\bbl@pictresetdir\relax
6874 \def\bbl@tempc{0}%
6875 \else
6876 \directlua{
6877 Babel.get_picture_dir = true
6878 Babel.picture_has_bidi = 0
6879 }%
6880 \setbox\z@\hb@xt@\z@{%
6881 \@defaultunitsset\@tempdimc{#1}\unitlength
6882 \kern\@tempdimc
6883 #3\hss}% TODO: #3 executed twice (below). That's bad.
6884 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6885 \fi
6886 % Do:
6887 \@defaultunitsset\@tempdimc{#2}\unitlength
6888 \raise\@tempdimc\hb@xt@\z@{%
6889 \@defaultunitsset\@tempdimc{#1}\unitlength
6890 \kern\@tempdimc
6891 {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6892 \ignorespaces}%
6893 \MakeRobust\put}%
6894 \AtBeginDocument
6895 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6896 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6897 \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6898 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6899 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z}%
6900 \fi
6901 \ifx\tikzpicture\undefined\else
6902 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
6903 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6904 \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
6905 \fi

```

```

6906 \ifx\tcolorbox\undefined\else
6907 \def\tcb@drawing@env@begin{%
6908 \csname tcb@before@\tcb@split@state\endcsname
6909 \bbl@pictsetdir\tw@
6910 \begin{\kvtcb@graphenv}%
6911 \tcb@bbdraw
6912 \tcb@apply@graph@patches}%
6913 \def\tcb@drawing@env@end{%
6914 \end{\kvtcb@graphenv}%
6915 \bbl@pictresetdir
6916 \csname tcb@after@\tcb@split@state\endcsname}%
6917 \fi
6918 }}
6919 {}

```

```

6920 \IfBabelLayout{counters*}%
6921   {\bbl@add\bbl@opt@layout{.counters.}%
6922     \directlua{
6923       luatexbase.add_to_callback("process_output_buffer",
6924         Babel.discard_subl_r , "Babel.discard_subl_r") }%
6925   }}
6926 \IfBabelLayout{counters}%
6927   {\let\bbl@0L@@@textsuperscript@textsuperscript
6928     \bbl@replace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6929     \let\bbl@latinarabic=@arabic
6930     \let\bbl@0L@@arabic@arabic
6931     \def@arabic#1{\babelsubl_r{\bbl@latinarabic#1}}%
6932     \@ifpackagewith{babel}{bidi=default}%
6933     {\let\bbl@asciroman=@roman
6934       \let\bbl@0L@@roman@roman
6935       \def@roman#1{\babelsubl_r{\ensureascii{\bbl@asciroman#1}}}%
6936       \let\bbl@asciiRoman=@Roman
6937       \let\bbl@0L@@roman@Roman
6938       \def@Roman#1{\babelsubl_r{\ensureascii{\bbl@asciiRoman#1}}}%
6939       \let\bbl@0L@labelenumii\labelenumii
6940       \def\labelenumii{}\theenumii}%
6941       \let\bbl@0L@p@enumiii\p@enumiii
6942       \def\p@enumiii{\p@enumii}\theenumii{}}{}{}{}
6943   }{Footnote changes}}
6944 \IfBabelLayout{footnotes}%
6945   {\let\bbl@0L@footnote\footnote
6946     \BabelFootnote\footnote\languagenamex{}{}}%
6947     \BabelFootnote\localfootnote\languagenamex{}{}}%
6948     \BabelFootnote\mainfootnote{}{}{}}
6949   {}

```

```

6950 \IfBabelLayout{extras}%
6951   {\bbl@ncarg\let\bbl@OL@underline{underline }%
6952     \bbl@carg\bbl@sreplace{underline }%
6953       {\$@@@underline}{\bgroup\bbl@nextfake$@@@underline}%
6954     \bbl@carg\bbl@sreplace{underline }%
6955       {\m@th$}{\m@th$\egroup}%
6956     \let\bbl@OL@LaTeXe\LaTeXe
6957     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6958       \if b\expandafter\car\@f@series\@nil\boldmath\fi
6959       \babelsublr{%
6960         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6961   {}
6962 \end{luatex}

```

## 10.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```
6963 (*transforms)
6964 Babel.linebreaking.replacements = {}
6965 Babel.linebreaking.replacements[0] = {} -- pre
6966 Babel.linebreaking.replacements[1] = {} -- post
6967
6968 -- Discretionaries contain strings as nodes
6969 function Babel.str_to_nodes(fn, matches, base)
6970   local n, head, last
6971   if fn == nil then return nil end
6972   for s in string.utfvalues(fn(matches)) do
6973     if base.id == 7 then
6974       base = base.replace
6975     end
6976     n = node.copy(base)
6977     n.char = s
6978     if not head then
6979       head = n
6980     else
6981       last.next = n
6982     end
6983     last = n
6984   end
6985   return head
6986 end
6987
6988 Babel.fetch_subtext = {}
6989
6990 Babel.ignore_pre_char = function(node)
6991   return (node.lang == Babel.nohyphenation)
6992 end
6993
6994 -- Merging both functions doesn't seem feasible, because there are too
6995 -- many differences.
6996 Babel.fetch_subtext[0] = function(head)
6997   local word_string = ''
6998   local word_nodes = {}
6999   local lang
7000   local item = head
7001   local inmath = false
7002
7003   while item do
7004     if item.id == 11 then
7005       inmath = (item.subtype == 0)
7006     end
7007
7008     if inmath then
7009       -- pass
7010     elseif item.id == 29 then
```

```

7013     local locale = node.get_attribute(item, Babel.attr_locale)
7014
7015     if lang == locale or lang == nil then
7016         lang = lang or locale
7017         if Babel.ignore_pre_char(item) then
7018             word_string = word_string .. Babel.us_char
7019         else
7020             word_string = word_string .. unicode.utf8.char(item.char)
7021         end
7022         word_nodes[#word_nodes+1] = item
7023     else
7024         break
7025     end
7026
7027     elseif item.id == 12 and item.subtype == 13 then
7028         word_string = word_string .. ' '
7029         word_nodes[#word_nodes+1] = item
7030
7031     -- Ignore leading unrecognized nodes, too.
7032     elseif word_string ~= '' then
7033         word_string = word_string .. Babel.us_char
7034         word_nodes[#word_nodes+1] = item -- Will be ignored
7035     end
7036
7037     item = item.next
7038 end
7039
7040 -- Here and above we remove some trailing chars but not the
7041 -- corresponding nodes. But they aren't accessed.
7042 if word_string:sub(-1) == ' ' then
7043     word_string = word_string:sub(1,-2)
7044 end
7045 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7046 return word_string, word_nodes, item, lang
7047 end
7048
7049 Babel.fetch_subtext[1] = function(head)
7050     local word_string = ''
7051     local word_nodes = {}
7052     local lang
7053     local item = head
7054     local inmath = false
7055
7056     while item do
7057
7058         if item.id == 11 then
7059             inmath = (item.subtype == 0)
7060         end
7061
7062         if inmath then
7063             -- pass
7064         end
7065
7066         elseif item.id == 29 then
7067             if item.lang == lang or lang == nil then
7068                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7069                     lang = lang or item.lang
7070                     word_string = word_string .. unicode.utf8.char(item.char)
7071                     word_nodes[#word_nodes+1] = item
7072                 end
7073             else
7074                 break
7075             end
7076         end
7077     end

```

```

7076     elseif item.id == 7 and item.subtype == 2 then
7077         word_string = word_string .. '='
7078         word_nodes[#word_nodes+1] = item
7079
7080     elseif item.id == 7 and item.subtype == 3 then
7081         word_string = word_string .. '|'
7082         word_nodes[#word_nodes+1] = item
7083
7084     -- (1) Go to next word if nothing was found, and (2) implicitly
7085     -- remove leading USs.
7086     elseif word_string == '' then
7087         -- pass
7088
7089     -- This is the responsible for splitting by words.
7090     elseif (item.id == 12 and item.subtype == 13) then
7091         break
7092
7093     else
7094         word_string = word_string .. Babel.us_char
7095         word_nodes[#word_nodes+1] = item -- Will be ignored
7096     end
7097
7098     item = item.next
7099 end
7100
7101 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7102 return word_string, word_nodes, item, lang
7103 end
7104
7105 function Babel.pre_hyphenate_replace(head)
7106     Babel.hyphenate_replace(head, 0)
7107 end
7108
7109 function Babel.post_hyphenate_replace(head)
7110     Babel.hyphenate_replace(head, 1)
7111 end
7112
7113 Babel.us_char = string.char(31)
7114
7115 function Babel.hyphenate_replace(head, mode)
7116     local u = unicode.utf8
7117     local lbkr = Babel.linebreaking.replacements[mode]
7118
7119     local word_head = head
7120
7121     while true do -- for each subtext block
7122
7123         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7124
7125         if Babel.debug then
7126             print()
7127             print((mode == 0) and '@@@<' or '@@@>', w)
7128         end
7129
7130         if nw == nil and w == '' then break end
7131
7132         if not lang then goto next end
7133         if not lbkr[lang] then goto next end
7134
7135         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7136         -- loops are nested.
7137         for k=1, #lbkr[lang] do
7138             local p = lbkr[lang][k].pattern

```

```

7139     local r = lbkr[lang][k].replace
7140     local attr = lbkr[lang][k].attr or -1
7141
7142     if Babel.debug then
7143         print('*****', p, mode)
7144     end
7145
7146     -- This variable is set in some cases below to the first *byte*
7147     -- after the match, either as found by u.match (faster) or the
7148     -- computed position based on sc if w has changed.
7149     local last_match = 0
7150     local step = 0
7151
7152     -- For every match.
7153     while true do
7154         if Babel.debug then
7155             print('====')
7156         end
7157         local new -- used when inserting and removing nodes
7158         local dummy_node -- used by after
7159
7160         local matches = { u.match(w, p, last_match) }
7161
7162         if #matches < 2 then break end
7163
7164         -- Get and remove empty captures (with ())'s, which return a
7165         -- number with the position), and keep actual captures
7166         -- (from (...)), if any, in matches.
7167         local first = table.remove(matches, 1)
7168         local last = table.remove(matches, #matches)
7169         -- Non re-fetched substrings may contain \31, which separates
7170         -- subsubstrings.
7171         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7172
7173         local save_last = last -- with A()BC()D, points to D
7174
7175         -- Fix offsets, from bytes to unicode. Explained above.
7176         first = u.len(w:sub(1, first-1)) + 1
7177         last = u.len(w:sub(1, last-1)) -- now last points to C
7178
7179         -- This loop stores in a small table the nodes
7180         -- corresponding to the pattern. Used by 'data' to provide a
7181         -- predictable behavior with 'insert' (w_nodes is modified on
7182         -- the fly), and also access to 'remove'd nodes.
7183         local sc = first-1 -- Used below, too
7184         local data_nodes = {}
7185
7186         local enabled = true
7187         for q = 1, last-first+1 do
7188             data_nodes[q] = w_nodes[sc+q]
7189             if enabled
7190                 and attr > -1
7191                 and not node.has_attribute(data_nodes[q], attr)
7192             then
7193                 enabled = false
7194             end
7195         end
7196
7197         -- This loop traverses the matched substring and takes the
7198         -- corresponding action stored in the replacement list.
7199         -- sc = the position in substr nodes / string
7200         -- rc = the replacement table index
7201         local rc = 0

```



```

7202
7203 ----- TODO. dummy_node?
7204 while rc < last-first+1 or dummy_node do -- for each replacement
7205     if Babel.debug then
7206         print('.....', rc + 1)
7207     end
7208     sc = sc + 1
7209     rc = rc + 1
7210
7211     if Babel.debug then
7212         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7213         local ss = ''
7214         for itt in node.traverse(head) do
7215             if itt.id == 29 then
7216                 ss = ss .. unicode.utf8.char(itt.char)
7217             else
7218                 ss = ss .. '{' .. itt.id .. '}'
7219             end
7220         end
7221         print('*****', ss)
7222
7223     end
7224
7225     local crep = r[rc]
7226     local item = w_nodes[sc]
7227     local item_base = item
7228     local placeholder = Babel.us_char
7229     local d
7230
7231     if crep and crep.data then
7232         item_base = data_nodes[crep.data]
7233     end
7234
7235     if crep then
7236         step = crep.step or step
7237     end
7238
7239     if crep and crep.after then
7240         crep.insert = true
7241         if dummy_node then
7242             item = dummy_node
7243         else -- TODO. if there is a node after?
7244             d = node.copy(item_base)
7245             head, item = node.insert_after(head, item, d)
7246             dummy_node = item
7247         end
7248     end
7249
7250     if crep and not crep.after and dummy_node then
7251         node.remove(head, dummy_node)
7252         dummy_node = nil
7253     end
7254
7255     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7256         if step == 0 then
7257             last_match = save_last -- Optimization
7258         else
7259             last_match = utf8.offset(w, sc+step)
7260         end
7261         goto next
7262
7263     elseif crep == nil or crep.remove then
7264         node.remove(head, item)

```

```

7265         table.remove(w_nodes, sc)
7266         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7267         sc = sc - 1 -- Nothing has been inserted.
7268         last_match = utf8.offset(w, sc+1+step)
7269         goto next
7270
7271     elseif crep and crep.kashida then -- Experimental
7272         node.set_attribute(item,
7273             Babel.attr_kashida,
7274             crep.kashida)
7275         last_match = utf8.offset(w, sc+1+step)
7276         goto next
7277
7278     elseif crep and crep.string then
7279         local str = crep.string(matches)
7280         if str == '' then -- Gather with nil
7281             node.remove(head, item)
7282             table.remove(w_nodes, sc)
7283             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7284             sc = sc - 1 -- Nothing has been inserted.
7285         else
7286             local loop_first = true
7287             for s in string.utfvalues(str) do
7288                 d = node.copy(item_base)
7289                 d.char = s
7290                 if loop_first then
7291                     loop_first = false
7292                     head, new = node.insert_before(head, item, d)
7293                     if sc == 1 then
7294                         word_head = head
7295                     end
7296                     w_nodes[sc] = d
7297                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7298                 else
7299                     sc = sc + 1
7300                     head, new = node.insert_before(head, item, d)
7301                     table.insert(w_nodes, sc, new)
7302                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7303                 end
7304                 if Babel.debug then
7305                     print('.....', 'str')
7306                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7307                 end
7308             end -- for
7309             node.remove(head, item)
7310         end -- if ''
7311         last_match = utf8.offset(w, sc+1+step)
7312         goto next
7313
7314     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7315         d = node.new(7, 3) -- (disc, regular)
7316         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7317         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7318         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7319         d.attr = item_base.attr
7320         if crep.pre == nil then -- TeXbook p96
7321             d.penalty = crep.penalty or tex.hyphenpenalty
7322         else
7323             d.penalty = crep.penalty or tex.exhyphenpenalty
7324         end
7325         placeholder = '|'
7326         head, new = node.insert_before(head, item, d)
7327

```

```

7328     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7329         -- ERROR
7330
7331     elseif crep and crep.penalty then
7332         d = node.new(14, 0) -- (penalty, userpenalty)
7333         d.attr = item_base.attr
7334         d.penalty = crep.penalty
7335         head, new = node.insert_before(head, item, d)
7336
7337     elseif crep and crep.space then
7338         -- 655360 = 10 pt = 10 * 65536 sp
7339         d = node.new(12, 13) -- (glue, spaceskip)
7340         local quad = font.getfont(item_base.font).size or 655360
7341         node.setglue(d, crep.space[1] * quad,
7342                       crep.space[2] * quad,
7343                       crep.space[3] * quad)
7344         if mode == 0 then
7345             placeholder = ' '
7346         end
7347         head, new = node.insert_before(head, item, d)
7348
7349     elseif crep and crep.norule then
7350         -- 655360 = 10 pt = 10 * 65536 sp
7351         d = node.new(2, 3) -- (rule, empty) = \no*rule
7352         local quad = font.getfont(item_base.font).size or 655360
7353         d.width = crep.norule[1] * quad
7354         d.height = crep.norule[2] * quad
7355         d.depth = crep.norule[3] * quad
7356         head, new = node.insert_before(head, item, d)
7357
7358     elseif crep and crep.spacefactor then
7359         d = node.new(12, 13) -- (glue, spaceskip)
7360         local base_font = font.getfont(item_base.font)
7361         node.setglue(d,
7362                     crep.spacefactor[1] * base_font.parameters['space'],
7363                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
7364                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
7365         if mode == 0 then
7366             placeholder = ' '
7367         end
7368         head, new = node.insert_before(head, item, d)
7369
7370     elseif mode == 0 and crep and crep.space then
7371         -- ERROR
7372
7373     elseif crep and crep.kern then
7374         d = node.new(13, 1) -- (kern, user)
7375         local quad = font.getfont(item_base.font).size or 655360
7376         d.attr = item_base.attr
7377         d.kern = crep.kern * quad
7378         head, new = node.insert_before(head, item, d)
7379
7380     elseif crep and crep.node then
7381         d = node.new(crep.node[1], crep.node[2])
7382         d.attr = item_base.attr
7383         head, new = node.insert_before(head, item, d)
7384
7385     end -- ie replacement cases
7386
7387     -- Shared by disc, space(factor), kern, node and penalty.
7388     if sc == 1 then
7389         word_head = head
7390     end

```

```

7391         if crep.insert then
7392             w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7393             table.insert(w_nodes, sc, new)
7394             last = last + 1
7395         else
7396             w_nodes[sc] = d
7397             node.remove(head, item)
7398             w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7399         end
7400
7401         last_match = utf8.offset(w, sc+1+step)
7402
7403         ::next::
7404
7405     end -- for each replacement
7406
7407     if Babel.debug then
7408         print('.....', '/')
7409         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7410     end
7411
7412     if dummy_node then
7413         node.remove(head, dummy_node)
7414         dummy_node = nil
7415     end
7416
7417     end -- for match
7418
7419 end -- for patterns
7420
7421 ::next::
7422 word_head = nw
7423 end -- for substring
7424 return head
7425 end
7426
7427 -- This table stores capture maps, numbered consecutively
7428 Babel.capture_maps = {}
7429
7430 -- The following functions belong to the next macro
7431 function Babel.capture_func(key, cap)
7432     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..[") .. "]"
7433     local cnt
7434     local u = unicode.utf8
7435     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7436     if cnt == 0 then
7437         ret = u.gsub(ret, '{(%x%x%x%x+)}',
7438             function (n)
7439                 return u.char(tonumber(n, 16))
7440             end)
7441     end
7442     ret = ret:gsub("%[%[%]]%.%", '')
7443     ret = ret:gsub("%.%.%[%[%]]%", '')
7444     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7445 end
7446
7447 function Babel.capt_map(from, mapno)
7448     return Babel.capture_maps[mapno][from] or from
7449 end
7450
7451 -- Handle the {n|abc|ABC} syntax in captures
7452 function Babel.capture_func_map(capno, from, to)
7453     local u = unicode.utf8

```

```

7454 from = u.gsub(from, '{(%x%x%x%x+)}',
7455               function (n)
7456                 return u.char(tonumber(n, 16))
7457               end)
7458 to = u.gsub(to, '{(%x%x%x%x+)}',
7459            function (n)
7460              return u.char(tonumber(n, 16))
7461            end)
7462 local froms = {}
7463 for s in string.utfcharacters(from) do
7464   table.insert(froms, s)
7465 end
7466 local cnt = 1
7467 table.insert(Babel.capture_maps, {})
7468 local mlen = table.getn(Babel.capture_maps)
7469 for s in string.utfcharacters(to) do
7470   Babel.capture_maps[mlen][froms[cnt]] = s
7471   cnt = cnt + 1
7472 end
7473 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7474        (mlen) .. ").." .. "["
7475 end
7476
7477 -- Create/Extend reversed sorted list of kashida weights:
7478 function Babel.capture_kashida(key, wt)
7479   wt = tonumber(wt)
7480   if Babel.kashida_wts then
7481     for p, q in ipairs(Babel.kashida_wts) do
7482       if wt == q then
7483         break
7484       elseif wt > q then
7485         table.insert(Babel.kashida_wts, p, wt)
7486         break
7487       elseif table.getn(Babel.kashida_wts) == p then
7488         table.insert(Babel.kashida_wts, wt)
7489       end
7490     end
7491   else
7492     Babel.kashida_wts = { wt }
7493   end
7494   return 'kashida = ' .. wt
7495 end
7496
7497 function Babel.capture_node(id, subtype)
7498   local sbt = 0
7499   for k, v in pairs(node.subtypes(id)) do
7500     if v == subtype then sbt = k end
7501   end
7502   return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7503 end
7504
7505 -- Experimental: applies prehyphenation transforms to a string (letters
7506 -- and spaces).
7507 function Babel.string_prehyphenation(str, locale)
7508   local n, head, last, res
7509   head = node.new(8, 0) -- dummy (hack just to start)
7510   last = head
7511   for s in string.utfvalues(str) do
7512     if s == 20 then
7513       n = node.new(12, 0)
7514     else
7515       n = node.new(29, 0)
7516     end
7517     n.char = s

```

```

7517     end
7518     node.set_attribute(n, Babel.attr_locale, locale)
7519     last.next = n
7520     last = n
7521 end
7522 head = Babel.hyphenate_replace(head, 0)
7523 res = ''
7524 for n in node.traverse(head) do
7525     if n.id == 12 then
7526         res = res .. ' '
7527     elseif n.id == 29 then
7528         res = res .. unicode.utf8.char(n.char)
7529     end
7530 end
7531 tex.print(res)
7532 end
7533 </transforms>

```

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

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

```

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

```

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

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

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.

```

7534 (*basic-r)
7535 Babel = Babel or {}
7536
7537 Babel.bidi_enabled = true
7538
7539 require('babel-data-bidi.lua')
7540

```

```

7541 local characters = Babel.characters
7542 local ranges = Babel.ranges
7543
7544 local DIR = node.id("dir")
7545
7546 local function dir_mark(head, from, to, outer)
7547   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7548   local d = node.new(DIR)
7549   d.dir = '+' .. dir
7550   node.insert_before(head, from, d)
7551   d = node.new(DIR)
7552   d.dir = '-' .. dir
7553   node.insert_after(head, to, d)
7554 end
7555
7556 function Babel.bidi(head, ispar)
7557   local first_n, last_n          -- first and last char with nums
7558   local last_es                 -- an auxiliary 'last' used with nums
7559   local first_d, last_d         -- first and last char in L/R block
7560   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):

```

7561   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7562   local strong_lr = (strong == 'l') and 'l' or 'r'
7563   local outer = strong
7564
7565   local new_dir = false
7566   local first_dir = false
7567   local inmath = false
7568
7569   local last_lr
7570
7571   local type_n = ''
7572
7573   for item in node.traverse(head) do
7574
7575     -- three cases: glyph, dir, otherwise
7576     if item.id == node.id'glyph'
7577       or (item.id == 7 and item.subtype == 2) then
7578
7579       local itemchar
7580       if item.id == 7 and item.subtype == 2 then
7581         itemchar = item.replace.char
7582       else
7583         itemchar = item.char
7584       end
7585       local chardata = characters[itemchar]
7586       dir = chardata and chardata.d or nil
7587       if not dir then
7588         for nn, et in ipairs(ranges) do
7589           if itemchar < et[1] then
7590             break
7591           elseif itemchar <= et[2] then
7592             dir = et[3]
7593             break
7594           end
7595         end
7596       end
7597       dir = dir or 'l'
7598       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).

```

7599     if new_dir then
7600         attr_dir = 0
7601         for at in node.traverse(item.attr) do
7602             if at.number == Babel.attr_dir then
7603                 attr_dir = at.value & 0x3
7604             end
7605         end
7606         if attr_dir == 1 then
7607             strong = 'r'
7608         elseif attr_dir == 2 then
7609             strong = 'al'
7610         else
7611             strong = 'l'
7612         end
7613         strong_lr = (strong == 'l') and 'l' or 'r'
7614         outer = strong_lr
7615         new_dir = false
7616     end
7617
7618     if dir == 'nsm' then dir = strong end -- W1

```

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

```

7619     dir_real = dir -- We need dir_real to set strong below
7620     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:

```

7621     if strong == 'al' then
7622         if dir == 'en' then dir = 'an' end -- W2
7623         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7624         strong_lr = 'r' -- W3
7625     end

```

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

```

7626     elseif item.id == node.id'dir' and not inmath then
7627         new_dir = true
7628         dir = nil
7629     elseif item.id == node.id'math' then
7630         inmath = (item.subtype == 0)
7631     else
7632         dir = nil -- Not a char
7633     end

```

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```

7634     if dir == 'en' or dir == 'an' or dir == 'et' then
7635         if dir ~= 'et' then
7636             type_n = dir
7637         end
7638         first_n = first_n or item
7639         last_n = last_es or item
7640         last_es = nil
7641     elseif dir == 'es' and last_n then -- W3+W6
7642         last_es = item
7643     elseif dir == 'cs' then -- it's right - do nothing
7644     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil

```



```

7645     if strong_lr == 'r' and type_n ~= '' then
7646         dir_mark(head, first_n, last_n, 'r')
7647     elseif strong_lr == 'l' and first_d and type_n == 'an' then
7648         dir_mark(head, first_n, last_n, 'r')
7649         dir_mark(head, first_d, last_d, outer)
7650         first_d, last_d = nil, nil
7651     elseif strong_lr == 'l' and type_n ~= '' then
7652         last_d = last_n
7653     end
7654     type_n = ''
7655     first_n, last_n = nil, nil
7656 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:

```

7657     if dir == 'l' or dir == 'r' then
7658         if dir ~= outer then
7659             first_d = first_d or item
7660             last_d = item
7661         elseif first_d and dir ~= strong_lr then
7662             dir_mark(head, first_d, last_d, outer)
7663             first_d, last_d = nil, nil
7664         end
7665     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.

```

7666     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7667         item.char = characters[item.char] and
7668             characters[item.char].m or item.char
7669     elseif (dir or new_dir) and last_lr ~= item then
7670         local mir = outer .. strong_lr .. (dir or outer)
7671         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7672             for ch in node.traverse(node.next(last_lr)) do
7673                 if ch == item then break end
7674                 if ch.id == node.id'glyph' and characters[ch.char] then
7675                     ch.char = characters[ch.char].m or ch.char
7676                 end
7677             end
7678         end
7679     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).

```

7680     if dir == 'l' or dir == 'r' then
7681         last_lr = item
7682         strong = dir_real          -- Don't search back - best save now
7683         strong_lr = (strong == 'l') and 'l' or 'r'
7684     elseif new_dir then
7685         last_lr = nil
7686     end
7687 end

```

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

```

7688     if last_lr and outer == 'r' then
7689         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7690             if characters[ch.char] then
7691                 ch.char = characters[ch.char].m or ch.char
7692             end

```

```

7693     end
7694 end
7695 if first_n then
7696     dir_mark(head, first_n, last_n, outer)
7697 end
7698 if first_d then
7699     dir_mark(head, first_d, last_d, outer)
7700 end

```

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

```

7701 return node.prev(head) or head
7702 end
7703 </basic-r>

```

And here the Lua code for bidi=basic:

```

7704 (*basic)
7705 Babel = Babel or {}
7706
7707 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7708
7709 Babel.fontmap = Babel.fontmap or {}
7710 Babel.fontmap[0] = {}      -- l
7711 Babel.fontmap[1] = {}      -- r
7712 Babel.fontmap[2] = {}      -- al/an
7713
7714 -- To cancel mirroring. Also OML, OMS, U?
7715 Babel.symbol_fonts = Babel.symbol_fonts or {}
7716 Babel.symbol_fonts[font.id('tenln')] = true
7717 Babel.symbol_fonts[font.id('tenlnw')] = true
7718 Babel.symbol_fonts[font.id('tencirc')] = true
7719 Babel.symbol_fonts[font.id('tencircw')] = true
7720
7721 Babel.bidi_enabled = true
7722 Babel.mirroring_enabled = true
7723
7724 require('babel-data-bidi.lua')
7725
7726 local characters = Babel.characters
7727 local ranges = Babel.ranges
7728
7729 local DIR = node.id('dir')
7730 local GLYPH = node.id('glyph')
7731
7732 local function insert_implicit(head, state, outer)
7733     local new_state = state
7734     if state.sim and state.eim and state.sim ~= state.eim then
7735         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7736         local d = node.new(DIR)
7737         d.dir = '+' .. dir
7738         node.insert_before(head, state.sim, d)
7739         local d = node.new(DIR)
7740         d.dir = '-' .. dir
7741         node.insert_after(head, state.eim, d)
7742     end
7743     new_state.sim, new_state.eim = nil, nil
7744     return head, new_state
7745 end
7746
7747 local function insert_numeric(head, state)
7748     local new
7749     local new_state = state
7750     if state.san and state.ean and state.san ~= state.ean then
7751         local d = node.new(DIR)

```

```

7752     d.dir = '+TLT'
7753     _, new = node.insert_before(head, state.san, d)
7754     if state.san == state.sim then state.sim = new end
7755     local d = node.new(DIR)
7756     d.dir = '-TLT'
7757     _, new = node.insert_after(head, state.ean, d)
7758     if state.ean == state.eim then state.eim = new end
7759 end
7760 new_state.san, new_state.ean = nil, nil
7761 return head, new_state
7762 end
7763
7764 local function glyph_not_symbol_font(node)
7765     if node.id == GLYPH then
7766         return not Babel.symbol_fonts[node.font]
7767     else
7768         return false
7769     end
7770 end
7771
7772 -- TODO - \hbox with an explicit dir can lead to wrong results
7773 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7774 -- was s made to improve the situation, but the problem is the 3-dir
7775 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7776 -- well.
7777
7778 function Babel.bidi(head, ispar, hdir)
7779     local d -- d is used mainly for computations in a loop
7780     local prev_d = ''
7781     local new_d = false
7782
7783     local nodes = {}
7784     local outer_first = nil
7785     local inmath = false
7786
7787     local glue_d = nil
7788     local glue_i = nil
7789
7790     local has_en = false
7791     local first_et = nil
7792
7793     local has_hyperlink = false
7794
7795     local ATDIR = Babel.attr_dir
7796     local attr_d
7797
7798     local save_outer
7799     local temp = node.get_attribute(head, ATDIR)
7800     if temp then
7801         temp = temp & 0x3
7802         save_outer = (temp == 0 and 'l') or
7803                     (temp == 1 and 'r') or
7804                     (temp == 2 and 'al')
7805     elseif ispar then -- Or error? Shouldn't happen
7806         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7807     else -- Or error? Shouldn't happen
7808         save_outer = ('TRT' == hdir) and 'r' or 'l'
7809     end
7810     -- when the callback is called, we are just _after_ the box,
7811     -- and the textdir is that of the surrounding text
7812     -- if not ispar and hdir ~= tex.textdir then
7813     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7814     -- end

```

```

7815 local outer = save_outer
7816 local last = outer
7817 -- 'al' is only taken into account in the first, current loop
7818 if save_outer == 'al' then save_outer = 'r' end
7819
7820 local fontmap = Babel.fontmap
7821
7822 for item in node.traverse(head) do
7823
7824     -- In what follows, #node is the last (previous) node, because the
7825     -- current one is not added until we start processing the neutrals.
7826
7827     -- three cases: glyph, dir, otherwise
7828     if glyph_not_symbol_font(item)
7829         or (item.id == 7 and item.subtype == 2) then
7830
7831         if node.get_attribute(item, ATDIR) == 128 then goto nextnode end
7832
7833         local d_font = nil
7834         local item_r
7835         if item.id == 7 and item.subtype == 2 then
7836             item_r = item.replace    -- automatic discs have just 1 glyph
7837         else
7838             item_r = item
7839         end
7840
7841         local chardata = characters[item_r.char]
7842         d = chardata and chardata.d or nil
7843         if not d or d == 'nsm' then
7844             for nn, et in ipairs(ranges) do
7845                 if item_r.char < et[1] then
7846                     break
7847                 elseif item_r.char <= et[2] then
7848                     if not d then d = et[3]
7849                     elseif d == 'nsm' then d_font = et[3]
7850                     end
7851                     break
7852                 end
7853             end
7854         end
7855         d = d or 'l'
7856
7857         -- A short 'pause' in bidi for mapfont
7858         d_font = d_font or d
7859         d_font = (d_font == 'l' and 0) or
7860             (d_font == 'nsm' and 0) or
7861             (d_font == 'r' and 1) or
7862             (d_font == 'al' and 2) or
7863             (d_font == 'an' and 2) or nil
7864         if d_font and fontmap and fontmap[d_font][item_r.font] then
7865             item_r.font = fontmap[d_font][item_r.font]
7866         end
7867
7868         if new_d then
7869             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7870             if inmath then
7871                 attr_d = 0
7872             else
7873                 attr_d = node.get_attribute(item, ATDIR)
7874                 attr_d = attr_d & 0x3
7875             end
7876             if attr_d == 1 then
7877                 outer_first = 'r'

```

```

7878         last = 'r'
7879     elseif attr_d == 2 then
7880         outer_first = 'r'
7881         last = 'al'
7882     else
7883         outer_first = 'l'
7884         last = 'l'
7885     end
7886     outer = last
7887     has_en = false
7888     first_et = nil
7889     new_d = false
7890 end
7891
7892 if glue_d then
7893     if (d == 'l' and 'l' or 'r') ~= glue_d then
7894         table.insert(nodes, {glue_i, 'on', nil})
7895     end
7896     glue_d = nil
7897     glue_i = nil
7898 end
7899
7900 elseif item.id == DIR then
7901     d = nil
7902
7903     if head ~= item then new_d = true end
7904
7905     elseif item.id == node.id'glue' and item.subtype == 13 then
7906         glue_d = d
7907         glue_i = item
7908         d = nil
7909
7910     elseif item.id == node.id'math' then
7911         inmath = (item.subtype == 0)
7912
7913     elseif item.id == 8 and item.subtype == 19 then
7914         has_hyperlink = true
7915
7916     else
7917         d = nil
7918     end
7919
7920     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7921     if last == 'al' and d == 'en' then
7922         d = 'an'          -- W3
7923     elseif last == 'al' and (d == 'et' or d == 'es') then
7924         d = 'on'          -- W6
7925     end
7926
7927     -- EN + CS/ES + EN      -- W4
7928     if d == 'en' and #nodes >= 2 then
7929         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7930             and nodes[#nodes-1][2] == 'en' then
7931             nodes[#nodes][2] = 'en'
7932         end
7933     end
7934
7935     -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7936     if d == 'an' and #nodes >= 2 then
7937         if (nodes[#nodes][2] == 'cs')
7938             and nodes[#nodes-1][2] == 'an' then
7939             nodes[#nodes][2] = 'an'
7940         end

```

```

7941 end
7942
7943 -- ET/EN -- W5 + W7->l / W6->on
7944 if d == 'et' then
7945     first_et = first_et or (#nodes + 1)
7946 elseif d == 'en' then
7947     has_en = true
7948     first_et = first_et or (#nodes + 1)
7949 elseif first_et then -- d may be nil here !
7950     if has_en then
7951         if last == 'l' then
7952             temp = 'l' -- W7
7953         else
7954             temp = 'en' -- W5
7955         end
7956     else
7957         temp = 'on' -- W6
7958     end
7959     for e = first_et, #nodes do
7960         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7961     end
7962     first_et = nil
7963     has_en = false
7964 end
7965
7966 -- Force mathdir in math if ON (currently works as expected only
7967 -- with 'l')
7968
7969 if inmath and d == 'on' then
7970     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7971 end
7972
7973 if d then
7974     if d == 'al' then
7975         d = 'r'
7976         last = 'al'
7977     elseif d == 'l' or d == 'r' then
7978         last = d
7979     end
7980     prev_d = d
7981     table.insert(nodes, {item, d, outer_first})
7982 end
7983
7984 node.set_attribute(item, ATDIR, 128)
7985 outer_first = nil
7986
7987 ::nextnode::
7988
7989 end -- for each node
7990
7991 -- TODO -- repeated here in case EN/ET is the last node. Find a
7992 -- better way of doing things:
7993 if first_et then -- dir may be nil here !
7994     if has_en then
7995         if last == 'l' then
7996             temp = 'l' -- W7
7997         else
7998             temp = 'en' -- W5
7999         end
8000     else
8001         temp = 'on' -- W6
8002     end
8003     for e = first_et, #nodes do

```

```

8004     if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8005     end
8006 end
8007
8008 -- dummy node, to close things
8009 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8010
8011 ----- NEUTRAL -----
8012
8013 outer = save_outer
8014 last = outer
8015
8016 local first_on = nil
8017
8018 for q = 1, #nodes do
8019     local item
8020
8021     local outer_first = nodes[q][3]
8022     outer = outer_first or outer
8023     last = outer_first or last
8024
8025     local d = nodes[q][2]
8026     if d == 'an' or d == 'en' then d = 'r' end
8027     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8028
8029     if d == 'on' then
8030         first_on = first_on or q
8031     elseif first_on then
8032         if last == d then
8033             temp = d
8034         else
8035             temp = outer
8036         end
8037         for r = first_on, q - 1 do
8038             nodes[r][2] = temp
8039             item = nodes[r][1] -- MIRRORING
8040             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8041                 and temp == 'r' and characters[item.char] then
8042                 local font_mode = ''
8043                 if item.font > 0 and font.fonts[item.font].properties then
8044                     font_mode = font.fonts[item.font].properties.mode
8045                 end
8046                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8047                     item.char = characters[item.char].m or item.char
8048                 end
8049             end
8050         end
8051         first_on = nil
8052     end
8053
8054     if d == 'r' or d == 'l' then last = d end
8055 end
8056
8057 ----- IMPLICIT, REORDER -----
8058
8059 outer = save_outer
8060 last = outer
8061
8062 local state = {}
8063 state.has_r = false
8064
8065 for q = 1, #nodes do
8066

```

```

8067     local item = nodes[q][1]
8068
8069     outer = nodes[q][3] or outer
8070
8071     local d = nodes[q][2]
8072
8073     if d == 'nsm' then d = last end          -- W1
8074     if d == 'en' then d = 'an' end
8075     local isdir = (d == 'r' or d == 'l')
8076
8077     if outer == 'l' and d == 'an' then
8078         state.san = state.san or item
8079         state.ean = item
8080     elseif state.san then
8081         head, state = insert_numeric(head, state)
8082     end
8083
8084     if outer == 'l' then
8085         if d == 'an' or d == 'r' then      -- im -> implicit
8086             if d == 'r' then state.has_r = true end
8087             state.sim = state.sim or item
8088             state.eim = item
8089         elseif d == 'l' and state.sim and state.has_r then
8090             head, state = insert_implicit(head, state, outer)
8091         elseif d == 'l' then
8092             state.sim, state.eim, state.has_r = nil, nil, false
8093         end
8094     else
8095         if d == 'an' or d == 'l' then
8096             if nodes[q][3] then -- nil except after an explicit dir
8097                 state.sim = item -- so we move sim 'inside' the group
8098             else
8099                 state.sim = state.sim or item
8100             end
8101             state.eim = item
8102         elseif d == 'r' and state.sim then
8103             head, state = insert_implicit(head, state, outer)
8104         elseif d == 'r' then
8105             state.sim, state.eim = nil, nil
8106         end
8107     end
8108
8109     if isdir then
8110         last = d          -- Don't search back - best save now
8111     elseif d == 'on' and state.san then
8112         state.san = state.san or item
8113         state.ean = item
8114     end
8115
8116 end
8117
8118 head = node.prev(head) or head
8119
8120 ----- FIX HYPERLINKS -----
8121
8122 if has_hyperlink then
8123     local flag, linking = 0, 0
8124     for item in node.traverse(head) do
8125         if item.id == DIR then
8126             if item.dir == '+TRT' or item.dir == '+TLT' then
8127                 flag = flag + 1
8128             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8129                 flag = flag - 1

```



```

8130         end
8131     elseif item.id == 8 and item.subtype == 19 then
8132         linking = flag
8133     elseif item.id == 8 and item.subtype == 20 then
8134         if linking > 0 then
8135             if item.prev.id == DIR and
8136                 (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8137                 d = node.new(DIR)
8138                 d.dir = item.prev.dir
8139                 node.remove(head, item.prev)
8140                 node.insert_after(head, item, d)
8141             end
8142         end
8143         linking = 0
8144     end
8145 end
8146 end
8147
8148 return head
8149 end
8150 -- Make sure anything is marked as 'bidi done' (including nodes inserted
8151 -- after the babel algorithm).
8152 function Babel.unset_atdir(head)
8153     local ATDIR = Babel.attr_dir
8154     local MATH = node.id'math'
8155     local DIR = node.id'dir'
8156     for item in node.traverse(head) do
8157         node.set_attribute(item, ATDIR, 128)
8158     end
8159     return head
8160 end
8161 </basic>

```

## 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.

```

8162 <nil>
8163 \ProvidesLanguage{nil}[\<date> v\<version>] Nil language]
8164 \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.

```

8165 \ifx\l@nil\@undefined
8166     \newlanguage\l@nil

```

```

8167 \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
8168 \let\bbl@elt\relax
8169 \edef\bbl@languages{% Add it to the list of languages
8170 \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
8171 \fi

```

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

```
8172 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

`\captionnil`

`\datenil`

```

8173 \let\captionnil\@empty
8174 \let\datenil\@empty

```

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

```

8175 \def\bbl@inidata@nil{%
8176 \bbl@elt{identification}{tag.ini}{und}%
8177 \bbl@elt{identification}{load.level}{0}%
8178 \bbl@elt{identification}{charset}{utf8}%
8179 \bbl@elt{identification}{version}{1.0}%
8180 \bbl@elt{identification}{date}{2022-05-16}%
8181 \bbl@elt{identification}{name.local}{nil}%
8182 \bbl@elt{identification}{name.english}{nil}%
8183 \bbl@elt{identification}{name.babel}{nil}%
8184 \bbl@elt{identification}{tag.bcp47}{und}%
8185 \bbl@elt{identification}{language.tag.bcp47}{und}%
8186 \bbl@elt{identification}{tag.opentype}{dflt}%
8187 \bbl@elt{identification}{script.name}{Latin}%
8188 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8189 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8190 \bbl@elt{identification}{level}{1}%
8191 \bbl@elt{identification}{encodings}{}%
8192 \bbl@elt{identification}{derivate}{no}}
8193 \@namedef{bbl@tbcp@nil}{und}
8194 \@namedef{bbl@lbcp@nil}{und}
8195 \@namedef{bbl@casing@nil}{und} % TODO
8196 \@namedef{bbl@lotf@nil}{dflt}
8197 \@namedef{bbl@elname@nil}{nil}
8198 \@namedef{bbl@lname@nil}{nil}
8199 \@namedef{bbl@esname@nil}{Latin}
8200 \@namedef{bbl@sname@nil}{Latin}
8201 \@namedef{bbl@sbc@nil}{Latn}
8202 \@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.

```

8203 \ldf@finish{nil}
8204 \</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.

```

8205 \<Compute Julian day> \equiv
8206 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8207 \def\bbl@cs@gregleap#1{%
8208 (\bbl@fpmmod{#1}{4} == 0) \&\&
8209 (!((\bbl@fpmmod{#1}{100} == 0) \&\& (\bbl@fpmmod{#1}{400} != 0)))}
8210 \def\bbl@cs@jd#1#2#3{% year, month, day

```

```

8211 \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8212 floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8213 floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
8214 ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
8215 <</Compute Julian day>>

```

### 13.1 Islamic

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

```

8216 (*ca-islamic)
8217 \ExplSyntaxOn
8218 <<Compute Julian day>>
8219 % == islamic (default)
8220 % Not yet implemented
8221 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar:

```

8222 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8223 ((#3 + ceil(29.5 * (#2 - 1)) +
8224 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8225 1948439.5) - 1) }
8226 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8227 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8228 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8229 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8230 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8231 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8232 \edef\bbl@tempa{%
8233 \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8234 \edef#5{%
8235 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8236 \edef#6{\fp_eval:n{
8237 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8238 \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).

```

8239 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8240 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8241 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8242 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8243 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8244 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8245 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8246 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8247 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8248 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8249 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8250 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8251 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8252 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8253 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8254 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8255 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8256 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8257 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8258 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8259 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8260 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8261 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8262 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%

```

```

8263 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8264 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8265 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8266 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8267 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8268 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8269 65401,65431,65460,65490,65520}
8270 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8271 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8272 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{0}}
8273 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
8274 \ifnum#2>2014 \ifnum#2<2038
8275 \bbl@afterfi\expandafter\@gobble
8276 \fi\fi
8277 {\bbl@error{year-out-range}{2014-2038}}}%
8278 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8279 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8280 \count@\@ne
8281 \bbl@foreach\bbl@cs@umalqura@data{%
8282 \advance\count@\@ne
8283 \ifnum#1>\bbl@tempd\else
8284 \edef\bbl@tempe{\the\count@}%
8285 \edef\bbl@tempb{##1}%
8286 \fi}%
8287 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8288 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8289 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8290 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8291 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8292 \ExplSyntaxOff
8293 \bbl@add\bbl@precalendar{%
8294 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8295 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8296 \bbl@replace\bbl@ld@calendar{+}{}}%
8297 \bbl@replace\bbl@ld@calendar{-}{}}%
8298 \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 `hebcsl.sty`

```

8299 (*ca-hebrew)
8300 \newcount\bbl@cntcommon
8301 \def\bbl@remainder#1#2#3{%
8302 #3=#1\relax
8303 \divide #3 by #2\relax
8304 \multiply #3 by -#2\relax
8305 \advance #3 by #1\relax}%
8306 \newif\ifbbl@divisible
8307 \def\bbl@checkifdivisible#1#2{%
8308 {\countdef\tmp=0
8309 \bbl@remainder{#1}{#2}{\tmp}%
8310 \ifnum \tmp=0
8311 \global\bbl@divisibletrue
8312 \else
8313 \global\bbl@divisiblefalse
8314 \fi}}
8315 \newif\ifbbl@gregleap
8316 \def\bbl@ifgregleap#1{%
8317 \bbl@checkifdivisible{#1}{4}%
8318 \ifbbl@divisible
8319 \bbl@checkifdivisible{#1}{100}%

```

```

8320     \ifbbl@divisible
8321         \bbl@checkifdivisible{#1}{400}%
8322     \ifbbl@divisible
8323         \bbl@gregleaptrue
8324     \else
8325         \bbl@gregleapfalse
8326     \fi
8327 \else
8328     \bbl@gregleaptrue
8329 \fi
8330 \else
8331     \bbl@gregleapfalse
8332 \fi
8333 \ifbbl@gregleap}
8334 \def\bbl@gregdayspriormonths#1#2#3{%
8335     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8336         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8337     \bbl@ifgregleap{#2}%
8338     \ifnum #1 > 2
8339         \advance #3 by 1
8340     \fi
8341 \fi
8342     \global\bbl@cntcommon=#3}%
8343     #3=\bbl@cntcommon}
8344 \def\bbl@gregdaysprioryears#1#2{%
8345     {\countdef\tmpc=4
8346     \countdef\tmpb=2
8347     \tmpb=#1\relax
8348     \advance \tmpb by -1
8349     \tmpc=\tmpb
8350     \multiply \tmpc by 365
8351     #2=\tmpc
8352     \tmpc=\tmpb
8353     \divide \tmpc by 4
8354     \advance #2 by \tmpc
8355     \tmpc=\tmpb
8356     \divide \tmpc by 100
8357     \advance #2 by -\tmpc
8358     \tmpc=\tmpb
8359     \divide \tmpc by 400
8360     \advance #2 by \tmpc
8361     \global\bbl@cntcommon=#2\relax}%
8362     #2=\bbl@cntcommon}
8363 \def\bbl@absfromgreg#1#2#3#4{%
8364     {\countdef\tmpd=0
8365     #4=#1\relax
8366     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8367     \advance #4 by \tmpd
8368     \bbl@gregdaysprioryears{#3}{\tmpd}%
8369     \advance #4 by \tmpd
8370     \global\bbl@cntcommon=#4\relax}%
8371     #4=\bbl@cntcommon}
8372 \newif\ifbbl@hebrleap
8373 \def\bbl@checkleaphebryear#1{%
8374     {\countdef\tmpa=0
8375     \countdef\tmpb=1
8376     \tmpa=#1\relax
8377     \multiply \tmpa by 7
8378     \advance \tmpa by 1
8379     \bbl@remainder{\tmpa}{19}{\tmpb}%
8380     \ifnum \tmpb < 7
8381         \global\bbl@hebrleaptrue
8382     \else

```

```

8383     \global\bbl@hebrleapfalse
8384     \fi}}
8385 \def\bbl@hebreleapsedmonths#1#2{%
8386     {\countdef\tmpa=0
8387     \countdef\tmpb=1
8388     \countdef\tmpc=2
8389     \tmpa=#1\relax
8390     \advance\tmpa by -1
8391     #2=\tmpa
8392     \divide #2 by 19
8393     \multiply #2 by 235
8394     \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8395     \tmpc=\tmpb
8396     \multiply\tmpb by 12
8397     \advance #2 by \tmpb
8398     \multiply\tmpc by 7
8399     \advance\tmpc by 1
8400     \divide\tmpc by 19
8401     \advance #2 by \tmpc
8402     \global\bbl@cntcommon=#2}%
8403     #2=\bbl@cntcommon}
8404 \def\bbl@hebreleapseddays#1#2{%
8405     {\countdef\tmpa=0
8406     \countdef\tmpb=1
8407     \countdef\tmpc=2
8408     \bbl@hebreleapsedmonths{#1}{#2}%
8409     \tmpa=#2\relax
8410     \multiply\tmpa by 13753
8411     \advance\tmpa by 5604
8412     \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8413     \divide\tmpa by 25920
8414     \multiply #2 by 29
8415     \advance #2 by 1
8416     \advance #2 by \tmpa
8417     \bbl@remainder{#2}{7}{\tmpa}%
8418     \ifnum\tmpc < 19440
8419         \ifnum\tmpc < 9924
8420             \else
8421                 \ifnum\tmpa=2
8422                     \bbl@checkleaphebrewyear{#1}% of a common year
8423                     \ifbbl@hebrleap
8424                         \else
8425                             \advance #2 by 1
8426                         \fi
8427                     \fi
8428                 \fi
8429                 \ifnum\tmpc < 16789
8430                     \else
8431                         \ifnum\tmpa=1
8432                             \advance #1 by -1
8433                             \bbl@checkleaphebrewyear{#1}% at the end of leap year
8434                             \ifbbl@hebrleap
8435                                 \advance #2 by 1
8436                             \fi
8437                         \fi
8438                     \fi
8439                     \else
8440                         \advance #2 by 1
8441                     \fi
8442                     \bbl@remainder{#2}{7}{\tmpa}%
8443                     \ifnum\tmpa=0
8444                         \advance #2 by 1
8445                     \else

```

```

8446     \ifnum \tmpa=3
8447         \advance #2 by 1
8448     \else
8449         \ifnum \tmpa=5
8450             \advance #2 by 1
8451         \fi
8452     \fi
8453 \fi
8454 \global\bbbl@cntcommon=#2\relax}%
8455 #2=\bbbl@cntcommon}
8456 \def\bbbl@daysinhebrewyear#1#2{%
8457     {\countdef\tmpe=12
8458     \bbbl@hebreleapseddays{#1}{\tmpe}%
8459     \advance #1 by 1
8460     \bbbl@hebreleapseddays{#1}{#2}%
8461     \advance #2 by -\tmpe
8462     \global\bbbl@cntcommon=#2}%
8463 #2=\bbbl@cntcommon}
8464 \def\bbbl@hebrdayspriormonths#1#2#3{%
8465     {\countdef\tmpf= 14
8466     #3=\ifcase #1\relax
8467         0 \or
8468         0 \or
8469         30 \or
8470         59 \or
8471         89 \or
8472         118 \or
8473         148 \or
8474         148 \or
8475         177 \or
8476         207 \or
8477         236 \or
8478         266 \or
8479         295 \or
8480         325 \or
8481         400
8482     \fi
8483     \bbbl@checkleaphebrewyear{#2}%
8484     \ifbbbl@hebrleap
8485         \ifnum #1 > 6
8486             \advance #3 by 30
8487         \fi
8488     \fi
8489     \bbbl@daysinhebrewyear{#2}{\tmpf}%
8490     \ifnum #1 > 3
8491         \ifnum \tmpf=353
8492             \advance #3 by -1
8493         \fi
8494         \ifnum \tmpf=383
8495             \advance #3 by -1
8496         \fi
8497     \fi
8498     \ifnum #1 > 2
8499         \ifnum \tmpf=355
8500             \advance #3 by 1
8501         \fi
8502         \ifnum \tmpf=385
8503             \advance #3 by 1
8504         \fi
8505     \fi
8506     \global\bbbl@cntcommon=#3\relax}%
8507 #3=\bbbl@cntcommon}
8508 \def\bbbl@absfromhebr#1#2#3#4{%

```

```

8509 {#4=#1\relax
8510 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8511 \advance #4 by #1\relax
8512 \bbl@hebreleapseddays{#3}{#1}%
8513 \advance #4 by #1\relax
8514 \advance #4 by -1373429
8515 \global\bbl@cntcommon=#4\relax}%
8516 #4=\bbl@cntcommon}
8517 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8518 {\countdef\tmpx= 17
8519 \countdef\tmpy= 18
8520 \countdef\tmpz= 19
8521 #6=#3\relax
8522 \global\advance #6 by 3761
8523 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8524 \tmpz=1 \tmpy=1
8525 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8526 \ifnum \tmpx > #4\relax
8527 \global\advance #6 by -1
8528 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8529 \fi
8530 \advance #4 by -\tmpx
8531 \advance #4 by 1
8532 #5=#4\relax
8533 \divide #5 by 30
8534 \loop
8535 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8536 \ifnum \tmpx < #4\relax
8537 \advance #5 by 1
8538 \tmpy=\tmpx
8539 \repeat
8540 \global\advance #5 by -1
8541 \global\advance #4 by -\tmpy}}
8542 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8543 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8544 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8545 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8546 \bbl@hebrfromgreg
8547 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8548 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8549 \edef#4{\the\bbl@hebryear}%
8550 \edef#5{\the\bbl@hebrmonth}%
8551 \edef#6{\the\bbl@hebrday}}
8552 /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 LPL 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).

```

8553 (*ca-persian)
8554 \ExplSyntaxOn
8555 <<Compute Julian day>>
8556 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8557 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8558 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8559 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8560 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8561 \bbl@afterfi\expandafter@gobble
8562 \fi\fi
8563 {\bbl@error{year-out-range}{2013-2050}{}}}%

```



```

8564 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8565 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8566 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8567 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8568 \ifnum\bbl@tempc<\bbl@tempb
8569 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8570 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8571 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8572 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8573 \fi
8574 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8575 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8576 \edef#5{\fp_eval:n{% set Jalali month
8577 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8578 \edef#6{\fp_eval:n{% set Jalali day
8579 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8580 \ExplSyntaxOff
8581 \</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.

```

8582 \<ca-coptic>
8583 \ExplSyntaxOn
8584 \<<Compute Julian day>>
8585 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8586 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8587 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8588 \edef#4{\fp_eval:n{%
8589 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8590 \edef\bbl@tempc{\fp_eval:n{%
8591 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8592 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8593 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8594 \ExplSyntaxOff
8595 \</ca-coptic>
8596 \<ca-ethiopic>
8597 \ExplSyntaxOn
8598 \<<Compute Julian day>>
8599 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8600 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8601 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8602 \edef#4{\fp_eval:n{%
8603 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8604 \edef\bbl@tempc{\fp_eval:n{%
8605 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8606 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8607 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8608 \ExplSyntaxOff
8609 \</ca-ethiopic>

```

### 13.5 Buddhist

That's very simple.

```

8610 \<ca-buddhist>
8611 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8612 \edef#4{\number\numexpr#1+543\relax}%
8613 \edef#5{#2}%
8614 \edef#6{#3}}
8615 \</ca-buddhist>
8616 %

```

```

8617% \subsection{Chinese}
8618%
8619% Brute force, with the Julian day of first day of each month. The
8620% table has been computed with the help of \textsf{python-lunardate} by
8621% Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8622% is 2015-2044.
8623%
8624% \begin{macrocode}
8625 (*ca-chinese)
8626 \ExplSyntaxOn
8627 <<Compute Julian day>>
8628 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
8629 \edef\bbl@tempd{\fp_eval:n{
8630 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8631 \count@ \z@
8632 \@tempcnta=2015
8633 \bbl@foreach\bbl@cs@chinese@data{%
8634 \ifnum##1>\bbl@tempd\else
8635 \advance\count@\@ne
8636 \ifnum\count@>12
8637 \count@\@ne
8638 \advance\@tempcnta\@ne\fi
8639 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8640 \ifin@
8641 \advance\count@\m@ne
8642 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8643 \else
8644 \edef\bbl@tempe{\the\count@}%
8645 \fi
8646 \edef\bbl@tempb{##1}%
8647 \fi}%
8648 \edef#4{\the\@tempcnta}%
8649 \edef#5{\bbl@tempe}%
8650 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8651 \def\bbl@cs@chinese@leap{%
8652 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8653 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8654 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8655 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8656 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8657 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8658 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8659 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8660 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8661 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8662 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8663 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8664 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8665 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8666 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8667 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8668 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8669 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8670 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8671 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8672 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8673 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8674 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8675 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8676 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8677 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8678 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8679 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%

```

```

8680 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8681 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8682 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8683 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8684 10896,10926,10956,10986,11015,11045,11074,11103}
8685 \ExplSyntaxOff
8686 </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`.

```

8687 <*bplain | blplain>
8688 \catcode`\{=1 % left brace is begin-group character
8689 \catcode`\}=2 % right brace is end-group character
8690 \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).

```

8691 \openin 0 hyphen.cfg
8692 \ifeof0
8693 \else
8694   \let\input

```

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

```

8695 \def\input #1 {%
8696   \let\input\input
8697   \a hyphen.cfg
8698   \let\input\input
8699 }
8700 \fi
8701 </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`.

```

8702 <bplain>\a plain.tex
8703 <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.

```

8704 <bplain>\def\fmtname{babel-plain}
8705 <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 $\text{\LaTeX}$ features

The file `babel.def` expects some definitions made in the  $\text{\LaTeX} 2_{\epsilon}$  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).

```
8706 <<(*Emulate LaTeX)>> ≡
8707 \def\@empty{}
8708 \def\loadlocalcfg#1{%
8709   \openin0#1.cfg
8710   \ifeof0
8711     \closein0
8712   \else
8713     \closein0
8714     {\immediate\write16{*****}%
8715      \immediate\write16{* Local config file #1.cfg used}%
8716      \immediate\write16{*}%
8717     }
8718     \input #1.cfg\relax
8719   \fi
8720 \endofldef}
```

## 14.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```
8721 \long\def\@firstofone#1{#1}
8722 \long\def\@firstoftwo#1#2{#1}
8723 \long\def\@secondoftwo#1#2{#2}
8724 \def\@nnil{\@nil}
8725 \def\@gobbletwo#1#2{}
8726 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8727 \def\@star@or@long#1{%
8728   \@ifstar
8729   {\let\l@ngrel@x\relax#1}%
8730   {\let\l@ngrel@x\long#1}}
8731 \let\l@ngrel@x\relax
8732 \def\@car#1#2\@nil{#1}
8733 \def\@cdr#1#2\@nil{#2}
8734 \let\@typeset@protect\relax
8735 \let\protected@edef\edef
8736 \long\def\@gobble#1{}
8737 \edef\@backslashchar{\expandafter\@gobble\string\}
8738 \def\strip@prefix#1>{}
8739 \def\g@addto@macro#1#2{{%
8740   \toks@{\expandafter{#1#2}%
8741   \xdef#1{\the\toks@}}}
8742 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8743 \def\@nameuse#1{\csname #1\endcsname}
8744 \def\@ifundefined#1{%
8745   \expandafter\ifx\csname#1\endcsname\relax
8746     \expandafter\@firstoftwo
8747   \else
8748     \expandafter\@secondoftwo
8749   \fi}
8750 \def\@expandtwoargs#1#2#3{%
8751   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8752 \def\zap@space#1 #2{%
8753   #1%
8754   \ifx#2\@empty\else\expandafter\zap@space\fi
8755   #2}
8756 \let\bbl@trace\@gobble
8757 \def\bbl@error#1{% Implicit #2#3#4}
```

```

8758 \begingroup
8759 \catcode`\=0 \catcode`\==12 \catcode`\`=12
8760 \catcode`\^^M=5 \catcode`\%=14
8761 \input errbabel.def
8762 \endgroup
8763 \bbl@error{#1}}
8764 \def\bbl@warning#1{%
8765 \begingroup
8766 \newlinechar=`^^J
8767 \def\{^^J(babel) }%
8768 \message{\#1}%
8769 \endgroup}
8770 \let\bbl@infowarn\bbl@warning
8771 \def\bbl@info#1{%
8772 \begingroup
8773 \newlinechar=`^^J
8774 \def\{^^J}%
8775 \wlog{#1}%
8776 \endgroup}

```

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

```

8777 \ifx\@preamblecmds\undefined
8778 \def\@preamblecmds{}
8779 \fi
8780 \def\@onlypreamble#1{%
8781 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8782 \@preamblecmds\do#1}}
8783 \@onlypreamble\@onlypreamble

```

Mimic  $\text{\LaTeX}$ 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

8784 \def\begin{document}{%
8785 \@begin{document}hook
8786 \global\let\@begin{document}hook\undefined
8787 \def\do##1{\global\let##1\undefined}%
8788 \@preamblecmds
8789 \global\let\do\noexpand}
8790 \ifx\@begin{document}hook\undefined
8791 \def\@begin{document}hook{}
8792 \fi
8793 \@onlypreamble\@begin{document}hook
8794 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

We also have to mimic  $\text{\LaTeX}$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endoflfd`.

```

8795 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8796 \@onlypreamble\AtEndOfPackage
8797 \def\@endoflfd{}
8798 \@onlypreamble\@endoflfd
8799 \let\bbl@afterlang\@empty
8800 \chardef\bbl@opt@hyphenmap\z@

```

$\text{\LaTeX}$  needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

8801 \catcode`\&=\z@
8802 \ifx&\if@filesw\undefined
8803 \expandafter\let\csname if@filesw\expandafter\endcsname
8804 \csname iffalse\endcsname
8805 \fi
8806 \catcode`\&=4

```

Mimic  $\text{\LaTeX}$ 's commands to define control sequences.

```

8807 \def\newcommand{\@star@or@long\new@command}
8808 \def\new@command#1{%
8809   \@testopt{\@newcommand#1}{}
8810 \def\@newcommand#1[#2]{%
8811   \@ifnextchar [{\@xargdef#1[#2]}%
8812     {\@argdef#1[#2]}}
8813 \long\def\@argdef#1[#2]#3{%
8814   \@yargdef#1\@ne{#2}{#3}}
8815 \long\def\@xargdef#1[#2][#3]#4{%
8816   \expandafter\def\expandafter#1\expandafter{%
8817     \expandafter\@protected@testopt\expandafter #1%
8818     \csname\string#1\expandafter\endcsname{#3}}}%
8819 \expandafter\@yargdef \csname\string#1\endcsname
8820 \tw@{#2}{#4}}
8821 \long\def\@yargdef#1#2#3{%
8822   \@tempcnta#3\relax
8823   \advance \@tempcnta \@ne
8824   \let\@hash@\relax
8825   \edef\reserved@a{\ifx#2\tw@ [\@hash@]\fi}%
8826   \@tempcntb #2%
8827   \@whilenum \@tempcntb <\@tempcnta
8828   \do{%
8829     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8830     \advance\@tempcntb \@ne}%
8831   \let\@hash@##%
8832   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8833 \def\providecommand{\@star@or@long\provide@command}
8834 \def\provide@command#1{%
8835   \begingroup
8836     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8837   \endgroup
8838   \expandafter\@ifundefined\@gtempa
8839     {\def\reserved@a{\new@command#1}}%
8840     {\let\reserved@a\relax
8841     \def\reserved@a{\new@command\reserved@a}}%
8842   \reserved@a}%

8843 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8844 \def\declare@robustcommand#1{%
8845   \edef\reserved@a{\string#1}%
8846   \def\reserved@b{#1}%
8847   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8848   \edef#1{%
8849     \ifx\reserved@a\reserved@b
8850       \noexpand\x@protect
8851       \noexpand#1%
8852     \fi
8853     \noexpand\protect
8854     \expandafter\noexpand\csname
8855       \expandafter\@gobble\string#1 \endcsname
8856   }%
8857   \expandafter\new@command\csname
8858     \expandafter\@gobble\string#1 \endcsname
8859 }
8860 \def\x@protect#1{%
8861   \ifx\protect\@typeset@protect\else
8862     \@x@protect#1%
8863   \fi
8864 }
8865 \catcode\&=\z@ % Trick to hide conditionals
8866 \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`.

```

8867 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8868 \catcode`\&=4
8869 \ifx\in@\@undefined
8870 \def\in@#1#2{%
8871   \def\in@@##1##2##3\in@@{%
8872     \ifx\in@@##2\in@false\else\in@true\fi}%
8873   \in@@#2#1\in@\in@@}
8874 \else
8875   \let\bbl@tempa\@empty
8876 \fi
8877 \bbl@tempa

```

$\text{\TeX}$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\text{\TeX}$  we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

8878 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\text{\TeX}$  macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\text{\TeX}$  but we need the macro to be defined as a no-op.

```

8879 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their  $\text{\TeX 2}_{\epsilon}$  versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```

8880 \ifx\@tempcnta\@undefined
8881   \csname newcount\endcsname\@tempcnta\relax
8882 \fi
8883 \ifx\@tempcntb\@undefined
8884   \csname newcount\endcsname\@tempcntb\relax
8885 \fi

```

To prevent wasting two counters in  $\text{\TeX}$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

8886 \ifx\bye\@undefined
8887   \advance\count10 by -2\relax
8888 \fi
8889 \ifx\@ifnextchar\@undefined
8890   \def\@ifnextchar#1#2#3{%
8891     \let\reserved@d=#1%
8892     \def\reserved@a{#2}\def\reserved@b{#3}%
8893     \futurelet\@let@token\@ifnch}
8894 \def\@ifnch{%
8895   \ifx\@let@token\@sptoken
8896     \let\reserved@c\@xifnch
8897   \else
8898     \ifx\@let@token\reserved@d
8899       \let\reserved@c\reserved@a
8900     \else
8901       \let\reserved@c\reserved@b
8902     \fi
8903   \fi
8904   \reserved@c}
8905 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
8906 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8907 \fi
8908 \def\@testopt#1#2{%
8909   \@ifnextchar[#{#1}{#2}}
8910 \def\@protected@testopt#1{%
8911   \ifx\protect\@typeset@protect
8912     \expandafter\@testopt

```

```

8913 \else
8914   \@x@protect#1%
8915 \fi}
8916 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8917   #2\relax}\fi}
8918 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8919   \else\expandafter\@gobble\fi{#1}}

```

## 14.4 Encoding related macros

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

```

8920 \def\DeclareTextCommand{%
8921   \@dec@text@cmd\providecommand
8922 }
8923 \def\ProvideTextCommand{%
8924   \@dec@text@cmd\providecommand
8925 }
8926 \def\DeclareTextSymbol#1#2#3{%
8927   \@dec@text@cmd\chardef#1{#2}#3\relax
8928 }
8929 \def\@dec@text@cmd#1#2#3{%
8930   \expandafter\def\expandafter#2%
8931     \expandafter{%
8932       \csname#3-cmd\expandafter\endcsname
8933       \expandafter#2%
8934       \csname#3\string#2\endcsname
8935     }%
8936 %   \let\@ifdefinable\@rc@ifdefinable
8937   \expandafter#1\csname#3\string#2\endcsname
8938 }
8939 \def\@current@cmd#1{%
8940   \ifx\protect\@typeset@protect\else
8941     \noexpand#1\expandafter\@gobble
8942   \fi
8943 }
8944 \def\@changed@cmd#1#2{%
8945   \ifx\protect\@typeset@protect
8946     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8947       \expandafter\ifx\csname ?\string#1\endcsname\relax
8948         \expandafter\def\csname ?\string#1\endcsname{%
8949           \@changed@x@err{#1}%
8950         }%
8951       \fi
8952       \global\expandafter\let
8953         \csname\cf@encoding\string#1\expandafter\endcsname
8954         \csname ?\string#1\endcsname
8955     \fi
8956     \csname\cf@encoding\string#1%
8957       \expandafter\endcsname
8958   \else
8959     \noexpand#1%
8960   \fi
8961 }
8962 \def\@changed@x@err#1{%
8963   \errhelp{Your command will be ignored, type <return> to proceed}%
8964   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8965 \def\DeclareTextCommandDefault#1{%
8966   \DeclareTextCommand#1?%
8967 }
8968 \def\ProvideTextCommandDefault#1{%
8969   \ProvideTextCommand#1?%
8970 }
8971 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd

```



```

8972 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8973 \def\DeclareTextAccent#1#2#3{%
8974   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8975 }
8976 \def\DeclareTextCompositeCommand#1#2#3#4{%
8977   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8978   \edef\reserved@b{\string##1}%
8979   \edef\reserved@c{%
8980     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8981   \ifx\reserved@b\reserved@c
8982     \expandafter\expandafter\expandafter\ifx
8983       \expandafter\@car\reserved@a\relax\relax\@nil
8984       \@text@composite
8985     \else
8986       \edef\reserved@b##1{%
8987         \def\expandafter\noexpand
8988           \csname#2\string#1\endcsname###1{%
8989             \noexpand\@text@composite
8990               \expandafter\noexpand\csname#2\string#1\endcsname
8991               ###1\noexpand\@empty\noexpand\@text@composite
8992               {##1}%
8993             }%
8994           }%
8995       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8996     \fi
8997     \expandafter\def\csname\expandafter\string\csname
8998       #2\endcsname\string#1-\string#3\endcsname{#4}
8999   \else
9000     \errhelp{Your command will be ignored, type <return> to proceed}%
9001     \errmessage{\string\DeclareTextCompositeCommand\space used on
9002       inappropriate command \protect#1}
9003   \fi
9004 }
9005 \def\@text@composite#1#2#3\@text@composite{%
9006   \expandafter\@text@composite@x
9007     \csname\string#1-\string#2\endcsname
9008 }
9009 \def\@text@composite@x#1#2{%
9010   \ifx#1\relax
9011     #2%
9012   \else
9013     #1%
9014   \fi
9015 }
9016 %
9017 \def\@strip@args#1:#2-#3\@strip@args{#2}
9018 \def\DeclareTextComposite#1#2#3#4{%
9019   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9020   \bgroup
9021     \lccode`\@=#4%
9022     \lowercase{%
9023       \egroup
9024       \reserved@a @%
9025     }%
9026 }
9027 %
9028 \def\UseTextSymbol#1#2{#2}
9029 \def\UseTextAccent#1#2#3{}
9030 \def\@use@text@encoding#1{}
9031 \def\DeclareTextSymbolDefault#1#2{%
9032   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9033 }
9034 \def\DeclareTextAccentDefault#1#2{%

```

```

9035 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9036 }
9037 \def\cf@encoding{OT1}

```

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

```

9038 \DeclareTextAccent{"}{OT1}{127}
9039 \DeclareTextAccent{'}{OT1}{19}
9040 \DeclareTextAccent{^}{OT1}{94}
9041 \DeclareTextAccent`}{OT1}{18}
9042 \DeclareTextAccent{~}{OT1}{126}

```

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

```

9043 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9044 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
9045 \DeclareTextSymbol{\textquoteleft}{OT1}{`'}
9046 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
9047 \DeclareTextSymbol{\i}{OT1}{16}
9048 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\TeX$ -control sequence `\scriptsize` to be available. Because plain  $\TeX$  doesn't have such a sophisticated font mechanism as  $\TeX$  has, we just `\let` it to `\sevenrm`.

```

9049 \ifx\scriptsize\@undefined
9050 \let\scriptsize\sevenrm
9051 \fi

```

And a few more “dummy” definitions.

```

9052 \def\language{english}%
9053 \let\bbl@opt@shorthands\@nnil
9054 \def\bbl@ifshorthand#1#2#3{#2}%
9055 \let\bbl@language@opts\@empty
9056 \let\bbl@ensureinfo\@gobble
9057 \let\bbl@provide@locale\relax
9058 \ifx\babeloptionstrings\@undefined
9059 \let\bbl@opt@strings\@nnil
9060 \else
9061 \let\bbl@opt@strings\babeloptionstrings
9062 \fi
9063 \def\BabelStringsDefault{generic}
9064 \def\bbl@tempa{normal}
9065 \ifx\babeloptionmath\bbl@tempa
9066 \def\bbl@mathnormal{\noexpand\textormath}
9067 \fi
9068 \def\AfterBabelLanguage#1#2{}
9069 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
9070 \let\bbl@afterlang\relax
9071 \def\bbl@opt@safe{BR}
9072 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
9073 \ifx\bbl@trace\@undefined\def\bbl@trace#1{\fi
9074 \expandafter\newif\csname ifbbl@single\endcsname
9075 \chardef\bbl@bidimode\z@
9076 <</Emulate LaTeX>>

```

A proxy file:

```

9077 <*\plain>
9078 \input babel.def
9079 </\plain>

```

## 15 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and

Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful. 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  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).