

# Babel

Code

Version 24.1.40082  
2024/02/04

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 bahavior</b>	<b>76</b>
5.1	Cross referencing macros . . . . .	78
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	109
10.4	Southeast Asian scripts	116
10.5	CJK line breaking	117
10.6	Arabic justification	119
10.7	Common stuff	123
10.8	Automatic fonts and ids switching	123
10.9	Bidi	129
10.10	Layout	131
10.11	Lua: transforms	139
10.12	Lua: Auto bidi with <code>basic</code> and <code>basic-r</code>	147
<b>11</b>	<b>Data for CJK</b>	<b>158</b>
<b>12</b>	<b>The ‘nil’ language</b>	<b>159</b>
<b>13</b>	<b>Calendars</b>	<b>160</b>
13.1	Islamic	160
13.2	Hebrew	161
13.3	Persian	166
13.4	Coptic and Ethiopic	166
13.5	Buddhist	167
<b>14</b>	<b>Support for Plain TeX (<code>plain.def</code>)</b>	<b>168</b>
14.1	Not renaming <code>hyphen.tex</code>	168
14.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features	169
14.3	General tools	169
14.4	Encoding related macros	173
<b>15</b>	<b>Acknowledgements</b>	<b>176</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  $\LaTeX$  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=24.1.40082>>
2 <<date=2024/02/04>>
```

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

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in babel.def and in babel.sty, which means in  $\LaTeX$  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\bbbl@info\@gobble
249 \let\bbbl@infowarn\@gobble
250 \let\bbbl@warning\@gobble}
251 {}
252 %
253 \def\AfterBabelLanguage#1{%
254 \global\expandafter\bbbl@add\csname#1.ldf-h@k\endcsname}%
```

If the format created a list of loaded languages (in \bbbl@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\bbbl@languages\@undefined\else
256 \begingroup
257 \catcode\^^I=12
258 \ifpackagewith{babel}{showlanguages}{%
259 \begingroup
260 \def\bbbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
261 \wlog{<*languages>}%
262 \bbbl@languages
263 \wlog{</languages>}%
264 \endgroup}{%
265 \endgroup
266 \def\bbbl@elt#1#2#3#4{%
267 \ifnum#2=\z@
268 \gdef\bbbl@nulllanguage{#1}%
269 \def\bbbl@elt##1##2##3##4{%
270 \fi}%
271 \bbbl@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 \bbbl@trace{Defining option 'base'}
274 \ifpackagewith{babel}{base}{%
275 \let\bbbl@onlyswitch\@empty
276 \let\bbbl@provide@locale\relax
277 \input babel.def
278 \let\bbbl@onlyswitch\@undefined
279 \ifx\directlua\@undefined
280 \DeclareOption*{\bbbl@patterns{\CurrentOption}}%
281 \else
282 \input luababel.def
283 \DeclareOption*{\bbbl@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`.

```

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}}

```

`\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).

```

611 \def\BabelContentsFiles{toc,lof,lot}
612 \def\bbl@set@language#1{% from selectlanguage, pop@
613   % The old buggy way. Preserved for compatibility.
614   \edef\language#1%
615   \ifnum\escapechar=\expandafter`\string#1\@empty
616     \else\string#1\@empty\fi}%
617   \ifcat\relax\noexpand#1%
618     \expandafter\ifx\csname date\language\endcsname\relax
619       \edef\language#1%
620       \let\localename\language
621     \else
622       \bbl@info{Using '\string\language' instead of 'language' is\\%
623         deprecated. If what you want is to use a\\%
624         macro containing the actual locale, make\\%
625         sure it does not not match any language.\\%
626         Reported}%
627       \ifx\scantokens\@undefined
628         \def\localename{??}%
629       \else
630         \scantokens\expandafter{\expandafter
631           \def\expandafter\localename\expandafter{\language}}%
632       \fi
633     \fi
634   \else
635     \def\localename{#1}% This one has the correct catcodes
636   \fi
637   \select@language{\language}%
638   % write to auxs
639   \expandafter\ifx\csname date\language\endcsname\relax\else
640     \if@filesw
641       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
642         \bbl@savelastskip
643         \protected@write\@auxout{\string\babel@aux{\bbl@auxname}}}%
644         \bbl@restorelastskip
645       \fi
646       \bbl@usehooks{write}%
647     \fi
648   \fi}
649 %

```

```

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

```

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

```

```

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

```

```

761 % reset selector name
762 \let\bbl@selectorname\@empty}

```

`otherlanguage (env.)` The `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

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

```

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

```

768 \long\def\endotherlanguage{%
769   \global\@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 \begingroup
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#1
814 % TODO. name@map here?
815 \bbl@provide@locale
816 \bbl@iflanguage\language#1{%
817 \let\bbl@select@type\@ne
818 \expandafter\bbl@switch\expandafter{\language#1}}

```

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 \expandafter\bbl@usehooks{patterns}{#{#1}{\bbl@tempa}}%
840 % > luatex
841 \ifundefined{bbl@hyphenation@}{\relax!
842 \begingroup
843 \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
844 \ifin@else
845 \expandafter\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 `\langhyphenmins` 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@#1.1df\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\languagetext\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  $\LaTeX 2_{\epsilon}$ , so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.  
Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```

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

```

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  $\LaTeX$  macros completely in case their definitions change (they have changed in the past). A macro named `\macro` will be saved new control sequences named `\org@macro`.

```

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

```

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 `\ifincsn` is available. If there is, the expansion will be more robust.

```
1317 \begingroup
1318 \bbl@ifunset{ifincsn}% 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   \ifincsn
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@activestru`), 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}
```

`\usesshorthands` 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\usesshorthands{%
1403   \ifstar\bb@usesesh@s{\bb@usesesh@x{}}
1404 \def\bb@usesesh@s#1{%
1405   \bb@usesesh@x
1406     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1407     {#1}}
1408 \def\bb@usesesh@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@tribute 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 name}%
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 name}%
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\bb1@allowhyphens j}
2077 \DeclareTextCommand{\IJ}{OT1}{%
2078 I\kern-0.02em\bb1@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{s{#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     \\bbl@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\xdef{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 % == main ==
2615 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2616 \let\language\bbbl@savelangname
2617 \chardef\localeid\bbbl@savelocaleid\relax
2618 \fi
2619 % == hyphenrules (apply if current) ==
2620 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2621 \ifnum\bbbl@savelocaleid=\localeid
2622 \language\@nameuse{l@\language}%
2623 \fi
2624 \fi}

```

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

```

2625 \def\bbbl@provide@new#1{%
2626 \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2627 \@namedef{extras#1}{}%
2628 \@namedef{noextras#1}{}%
2629 \bbbl@startcommands*{#1}{captions}%
2630 \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2631 \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2632 \ifx##1\@empty\else
2633 \bbbl@exp{%
2634 \\\SetString\\##1{%
2635 \\\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2636 \expandafter\bbbl@tempb
2637 \fi}%
2638 \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2639 \else
2640 \ifx\bbbl@initoload\relax
2641 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2642 \else
2643 \bbbl@read@ini{\bbbl@initoload}2% % Same
2644 \fi
2645 \fi
2646 \StartBabelCommands*{#1}{date}%
2647 \ifx\bbbl@KVP@date\@nnil
2648 \bbbl@exp{%
2649 \\\SetString\\\today{\bbbl@nocaption{today}{#1today}}}%
2650 \else
2651 \bbbl@savetoday
2652 \bbbl@savestate
2653 \fi
2654 \bbbl@endcommands
2655 \bbbl@load@basic{#1}%
2656 % == hyphenmins == (only if new)
2657 \bbbl@exp{%
2658 \gdef\<#1hyphenmins>{%
2659 {\bbbl@ifunset{\bbbl@lfthm#1}{2}{\bbbl@cs{lfthm#1}}}%
2660 {\bbbl@ifunset{\bbbl@rgthm#1}{3}{\bbbl@cs{rgthm#1}}}}}%
2661 % == hyphenrules (also in renew) ==
2662 \bbbl@provide@hyphens{#1}%
2663 \ifx\bbbl@KVP@main\@nnil\else
2664 \expandafter\main@language\expandafter{#1}%
2665 \fi}
2666 %
2667 \def\bbbl@provide@renew#1{%
2668 \ifx\bbbl@KVP@captions\@nnil\else
2669 \StartBabelCommands*{#1}{captions}%
2670 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11

```

```

2671 \EndBabelCommands
2672 \fi
2673 \ifx\bbk@KVP@date\@nnil\else
2674 \StartBabelCommands*{#1}{date}%
2675 \bbk@savetoday
2676 \bbk@savedate
2677 \EndBabelCommands
2678 \fi
2679 % == hyphenrules (also in new) ==
2680 \ifx\bbk@lbkflag\@empty
2681 \bbk@provide@hyphens{#1}%
2682 \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.)

```

2683 \def\bbk@load@basic#1{%
2684 \ifcase\bbk@howloaded\or\or
2685 \ifcase\csname bbl@llevel@\language\endcsname
2686 \bbk@csarg\let\lname@\language\relax
2687 \fi
2688 \fi
2689 \bbk@ifunset{bbl@lname@#1}%
2690 {\def\BabelBeforeIni##1##2{%
2691 \begingroup
2692 \let\bbk@ini@captions@aux\@gobbletwo
2693 \def\bbk@inidate ####1.####2.####3.####4\relax ####5####6}%
2694 \bbk@read@ini{##1}l%
2695 \ifx\bbk@initoload\relax\endinput\fi
2696 \endgroup}%
2697 \begingroup % boxed, to avoid extra spaces:
2698 \ifx\bbk@initoload\relax
2699 \bbk@input@texini{#1}%
2700 \else
2701 \setbox\z@\hbox{\BabelBeforeIni{\bbk@initoload}}}%
2702 \fi
2703 \endgroup}%
2704 {}}

```

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.

```

2705 \def\bbk@provide@hyphens#1{%
2706 \@tempcnta\m@ne % a flag
2707 \ifx\bbk@KVP@hyphenrules\@nnil\else
2708 \bbk@replace\bbk@KVP@hyphenrules{ }{,}%
2709 \bbk@foreach\bbk@KVP@hyphenrules{%
2710 \ifnum\@tempcnta=\m@ne % if not yet found
2711 \bbk@ifsamestring{##1}{+}%
2712 {\bbk@carg\addlanguage{l@##1}}%
2713 {}}%
2714 \bbk@ifunset{l@##1}% After a possible +
2715 {}%
2716 {\@tempcnta\@nameuse{l@##1}}%
2717 \fi}%
2718 \ifnum\@tempcnta=\m@ne
2719 \bbk@warning{%
2720 Requested 'hyphenrules' for '\language' not found:\\%
2721 \bbk@KVP@hyphenrules.\\%
2722 Using the default value. Reported}%
2723 \fi
2724 \fi
2725 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2726 \ifx\bbk@KVP@captions@\@nnil % TODO. Hackish. See above.
2727 \bbk@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists

```

```

2728      {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2729      }%
2730      {\bbl@ifunset{l@#1}{\bbl@cl{hyphr}}}%
2731      }%
2732      {\@tempcnta\@nameuse{l@#1}{\bbl@cl{hyphr}}}%
2733  \fi
2734 \fi
2735 \bbl@ifunset{l@#1}%
2736   {\ifnum\@tempcnta=\m@ne
2737    \bbl@carg\adddialect{l@#1}\language
2738    \else
2739    \bbl@carg\adddialect{l@#1}\@tempcnta
2740    \fi}%
2741   {\ifnum\@tempcnta=\m@ne\else
2742    \global\bbl@carg\chardef{l@#1}\@tempcnta
2743    \fi}}

```

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

```

2744 \def\bbl@input@texini#1{%
2745   \bbl@bsphack
2746   \bbl@exp{%
2747     \catcode`\\=14 \catcode`\\=0
2748     \catcode`\\={1 \catcode`\\}=2
2749     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2750     \catcode`\\=\the\catcode`\relax
2751     \catcode`\\=\the\catcode`\relax
2752     \catcode`\\=\the\catcode`\relax
2753     \catcode`\\=\the\catcode`\relax}%
2754   \bbl@esphack}

```

The following macros read and store ini files (but don't process them). For each line, there are 3 possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbl@read@ini.

```

2755 \def\bbl@iniline#1\bbl@iniline{%
2756   \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2757 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2758 \def\bbl@iniskip#1\@@{%
2759   \def\bbl@inistore#1=#2\@@%
2760   \bbl@trim@def\bbl@tempa{#1}%
2761   \bbl@trim\toks@{#2}%
2762   \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2763   \ifin@
2764     \bbl@xin@{,identification/include.}%
2765     {\bbl@section/\bbl@tempa}%
2766     \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2767     \bbl@exp{%
2768       \\g@addto@macro\\bbl@inidata{%
2769         \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2770   \fi}
2771 \def\bbl@inistore@min#1=#2\@@{%
2772   \bbl@trim@def\bbl@tempa{#1}%
2773   \bbl@trim\toks@{#2}%
2774   \bbl@xin@{.identification.}{\bbl@section.}%
2775   \ifin@
2776     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2777       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2778   \fi}

```

Now, the 'main loop', which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

```

2779 \def\bbl@loop@ini{%
2780   \loop
2781     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2782     \endlinechar\m@ne
2783     \read\bbl@readstream to \bbl@line
2784     \endlinechar\^^M
2785     \ifx\bbl@line\empty\else
2786       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2787     \fi
2788   \repeat}
2789 \ifx\bbl@readstream\undefined
2790   \csname newread\endcsname\bbl@readstream
2791 \fi
2792 \def\bbl@read@ini#1#2{%
2793   \global\let\bbl@extend@ini\@gobble
2794   \openin\bbl@readstream=babel-#1.ini
2795   \ifeof\bbl@readstream
2796     \bbl@error{no-ini-file}{#1}{}}%
2797   \else
2798     % == Store ini data in \bbl@inidata ==
2799     \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
2800     \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2801     \bbl@info{Importing
2802               \ifcase#2font and identification \or basic \fi
2803               data for \language\}%
2804     from babel-#1.ini. Reported}%
2805     \ifnum#2=\z@
2806       \global\let\bbl@inidata\empty
2807       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2808     \fi
2809     \def\bbl@section{identification}%
2810     \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
2811     \bbl@inistore load.level=#2\\@@
2812     \bbl@loop@ini
2813     % == Process stored data ==
2814     \bbl@csarg\xdef{lini@\language}{#1}%
2815     \bbl@read@ini@aux
2816     % == 'Export' data ==
2817     \bbl@ini@exports{#2}%
2818     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2819     \global\let\bbl@inidata\empty
2820     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
2821     \bbl@tglobal\bbl@ini@loaded
2822   \fi
2823   \closein\bbl@readstream}
2824 \def\bbl@read@ini@aux{%
2825   \let\bbl@savestrings\empty
2826   \let\bbl@savetoday\empty
2827   \let\bbl@savestate\empty
2828   \def\bbl@elt##1##2##3{%
2829     \def\bbl@section{##1}%
2830     \in{=date.}{=##1}% Find a better place
2831     \ifin@
2832       \bbl@ifunset{bbl@inikv{##1}}%
2833       {\bbl@ini@calendar{##1}}%
2834     {}%
2835   \fi
2836   \bbl@ifunset{bbl@inikv{##1}}{%
2837     {\csname bbl@inikv{##1}\endcsname{##2}{##3}}}%
2838   \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.



```

2839 \def\bb@extend@ini@aux#1{%
2840   \bb@startcommands*{#1}{captions}%
2841   % Activate captions/... and modify exports
2842   \bb@csarg\def\inikv@captions.licr{##1##2}%
2843   \setlocalecaption{#1}{##1}{##2}%
2844   \def\bb@inikv@captions##1##2{%
2845     \bb@ini@captions@aux{##1}{##2}%
2846     \def\bb@stringdef##1##2{\gdef##1{##2}}%
2847     \def\bb@exportkey##1##2##3{%
2848       \bb@ifunset{\bb@kv@##2}{}%
2849       {\expandafter\ifx\csname \bb@kv@##2\endcsname\@empty\else
2850         \bb@exp{\global\let\<\bb@##1@language\>\<\bb@kv@##2>}%
2851         \fi}}%
2852   % As with \bb@read@ini, but with some changes
2853   \bb@read@ini@aux
2854   \bb@ini@exports\tw@
2855   % Update inidata@lang by pretending the ini is read.
2856   \def\bb@elt##1##2##3{%
2857     \def\bb@section{##1}%
2858     \bb@iniline##2=##3\bb@iniline}%
2859     \csname \bb@inidata@#1\endcsname
2860     \global\bb@csarg\let\inidata@#1\bb@inidata
2861   \StartBabelCommands*{#1}{date}% And from the import stuff
2862   \def\bb@stringdef##1##2{\gdef##1{##2}}%
2863   \bb@savetoday
2864   \bb@savestate
2865   \bb@endcommands}

```

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

```

2866 \def\bb@ini@calendar#1{%
2867   \lowercase{\def\bb@tempa{=##1=}}%
2868   \bb@replace\bb@tempa{=date.gregorian}{}%
2869   \bb@replace\bb@tempa{=date.}{}%
2870   \in@{.licr}{=##1=}%
2871   \ifin@
2872     \ifcase\bb@engine
2873       \bb@replace\bb@tempa{.licr}{}%
2874     \else
2875       \let\bb@tempa\relax
2876     \fi
2877   \fi
2878   \ifx\bb@tempa\relax\else
2879     \bb@replace\bb@tempa{=}{}%
2880     \ifx\bb@tempa\@empty\else
2881       \xdef\bb@calendars{\bb@calendars,\bb@tempa}%
2882     \fi
2883     \bb@exp{%
2884       \def\<\bb@inikv@#1>####1####2{%
2885         \\\bb@inidata####1...\relax{####2}{\bb@tempa}}%
2886       \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 \bb@inistore above).

```

2887 \def\bb@renewinikey#1/#2\@#3{%
2888   \edef\bb@tempa{\zap@space #1 \@empty}% section
2889   \edef\bb@tempb{\zap@space #2 \@empty}% key
2890   \bb@trim\toks@{#3}% value
2891   \bb@exp{%
2892     \edef\\bb@key@list{\bb@key@list \bb@tempa/\bb@tempb;}%
2893     \\g@addto@macro\\bb@inidata{%
2894       \\bb@elt{\bb@tempa}{\bb@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.

```

2895 \def\bbl@exportkey#1#2#3{%
2896   \bbl@ifunset{\bbl@kv@#2}%
2897     {\bbl@csarg\gdef{#1@{\language\name}{#3}}}%
2898     {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2899       \bbl@csarg\gdef{#1@{\language\name}{#3}}%
2900       \else
2901         \bbl@exp{\global\let<\bbl@#1@{\language\name}>\<\bbl@kv@#2>}%
2902         \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.

```

2903 \def\bbl@iniwarning#1{%
2904   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2905   {\bbl@warning{%
2906     From babel-\bbl@cs{lini@{\language\name}}.ini:\%
2907     \bbl@cs{@kv@identification.warning#1}\%
2908     Reported }}%
2909 %
2910 \let\bbl@release@transforms\@empty
2911 \let\bbl@release@casing\@empty
2912 \def\bbl@ini@exports#1{%
2913   % Identification always exported
2914   \bbl@iniwarning{%
2915     \ifcase\bbl@engine
2916       \bbl@iniwarning{.pdf\latex}%
2917       \or
2918       \bbl@iniwarning{.lua\latex}%
2919       \or
2920       \bbl@iniwarning{.xel\latex}%
2921     \fi%
2922     \bbl@exportkey{llevel}{identification.load.level}{}%
2923     \bbl@exportkey{elname}{identification.name.english}{}%
2924     \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2925       {\csname bbl@elname@{\language\name}\endcsname}}%
2926     \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2927     % Somewhat hackish. TODO:
2928     \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2929     \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2930     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2931     \bbl@exportkey{esname}{identification.script.name}{}%
2932     \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2933       {\csname bbl@esname@{\language\name}\endcsname}}%
2934     \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2935     \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2936     \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2937     \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2938     \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2939     \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2940     \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2941     % Also maps bcp47 -> language\name
2942     \ifbbl@bcptoname
2943       \bbl@csarg\xdef{bcp@map@{\bbl@cl{tbc}}}{\language\name}%
2944     \fi
2945     \ifcase\bbl@engine\or
2946       \directlua{%
2947         Babel.locale_props[\the\bbl@cs{id@{\language\name}}].script
2948         = '\bbl@cl{sbc}}}%

```

```

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

```

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

```

2972 \def\bbl@inikv#1#2{%      key=value
2973 \toks@{#2}%              This hides #'s from ini values
2974 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2975 \let\bbl@inikv@identification\bbl@inikv
2976 \let\bbl@inikv@date\bbl@inikv
2977 \let\bbl@inikv@typography\bbl@inikv
2978 \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.

```

2979 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2980 \def\bbl@inikv@characters#1#2{%
2981 \bbl@ifsamestring{#1}{casing}% eg, casing = uV
2982 {\bbl@exp{%
2983 \\\g@addto@macro{\bbl@release@casing{%
2984 \\\bbl@casemapping}{\languagename}{\unexpanded{#2}}}}}%
2985 {\in@{$casing.}{$#1}% eg, casing.Uv = uV
2986 \ifin@
2987 \lowercase{\def\bbl@tempb{#1}}%
2988 \bbl@replace\bbl@tempb{casing.}{}%
2989 \bbl@exp{\\\g@addto@macro{\bbl@release@casing{%
2990 \\\bbl@casemapping
2991 {\\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}}}%
2992 \else
2993 \bbl@inikv{#1}{#2}%
2994 \fi}}

```

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

```

2995 \def\bbl@inikv@counters#1#2{%
2996 \bbl@ifsamestring{#1}{digits}%
2997 {\bbl@error{digits-is-reserved}{}}}%
2998 {}%
2999 \def\bbl@tempc{#1}%
3000 \bbl@trim@def{\bbl@tempc*}{#2}%

```

```

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

```

3016 \ifcase\bbl@engine
3017   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3018     \bbl@ini@captions@aux{#1}{#2}}
3019 \else
3020   \def\bbl@inikv@captions#1#2{%
3021     \bbl@ini@captions@aux{#1}{#2}}
3022 \fi

```

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

```

3023 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3024   \bbl@replace\bbl@tempa{.template}{}%
3025   \def\bbl@toreplace{#1}{}%
3026   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3027   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3028   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3029   \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}%
3030   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3031   \bbl@xin@{, \bbl@tempa, }, {chapter, appendix, part,}%
3032 \ifin@
3033   \@nameuse{\bbl@patch\bbl@tempa}%
3034   \global\bbl@csarg\let\bbl@tempa \bbl@toreplace
3035 \fi
3036 \bbl@xin@{, \bbl@tempa, }, {figure, table,}%
3037 \ifin@
3038   \global\bbl@csarg\let\bbl@tempa \bbl@toreplace
3039   \bbl@exp{\gdef\<fnum\bbl@tempa>{%
3040     \bbl@ifunset\bbl@tempa \bbl@tempa \bbl@tempa \bbl@tempa \bbl@tempa}%
3041     {\fnum\bbl@tempa}}%
3042     {\@nameuse{\bbl@tempa \bbl@tempa \bbl@tempa \bbl@tempa \bbl@tempa}}}%
3043 \fi}
3044 \def\bbl@ini@captions@aux#1#2{%
3045   \bbl@trim@def\bbl@tempa{#1}%
3046   \bbl@xin@{.template}{\bbl@tempa}%
3047 \ifin@
3048   \bbl@ini@captions@template{#2}\language
3049 \else
3050   \bbl@ifblank{#2}%
3051     {\bbl@exp{%
3052       \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3053     {\bbl@trim\toks@{#2}}}%
3054   \bbl@exp{%
3055     \bbl@add\bbl@savestrings{%
3056       \SetString\<\bbl@tempa name>{\the\toks@}}}%
3057   \toks@\expandafter\bbl@captionslist}%
3058   \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%

```

```

3059 \ifin@else
3060 \bbl@exp{%
3061 \\\bbl@add\<bbl@extracaps@\language\>\<\bbl@tempa name\>%
3062 \\\bbl@tglobal\<bbl@extracaps@\language\>%
3063 \fi
3064 \fi}

```

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

```

3065 \def\bbl@list@the{%
3066 part,chapter,section,subsection,subsubsection,paragraph,%
3067 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3068 table,page,footnote,mpfootnote,mpfn}
3069 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3070 \bbl@ifunset{bbl@map@#1@\language\>%
3071 {\@nameuse{#1}}%
3072 {\@nameuse{bbl@map@#1@\language\>}}}
3073 \def\bbl@inikv@labels#1#2{%
3074 \in@{.map}{#1}%
3075 \ifin@
3076 \ifx\bbl@KVP@labels\@nnil\else
3077 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3078 \ifin@
3079 \def\bbl@tempc{#1}%
3080 \bbl@replace\bbl@tempc{.map}{}%
3081 \in@{,#2,},{arabic,roman,Roman,alph,Alph,fnsymbol,}%
3082 \bbl@exp{%
3083 \gdef\<bbl@map@\bbl@tempc @\language\>%
3084 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3085 \bbl@foreach\bbl@list@the{%
3086 \bbl@ifunset{the##1}{}%
3087 {\bbl@exp{\let\\bbl@tempd\<the##1>%
3088 \bbl@exp{%
3089 \\\bbl@sreplace\<the##1>%
3090 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3091 \\\bbl@sreplace\<the##1>%
3092 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3093 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3094 \toks@expandafter\expandafter\expandafter{%
3095 \csname the##1\endcsname}%
3096 \expandafter\xdef\csname the##1\endcsname{\{the\toks@\}}%
3097 \fi}}%
3098 \fi
3099 \fi
3100 %
3101 \else
3102 %
3103 % The following code is still under study. You can test it and make
3104 % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3105 % language dependent.
3106 \in@{enumerate.}{#1}%
3107 \ifin@
3108 \def\bbl@tempa{#1}%
3109 \bbl@replace\bbl@tempa{enumerate.}{}%
3110 \def\bbl@toreplace{#2}%
3111 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3112 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3113 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}%
3114 \toks@expandafter{\bbl@toreplace}%
3115 % TODO. Execute only once:
3116 \bbl@exp{%
3117 \\\bbl@add\<extras\language\>%
3118 \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3119 \def\<labelenum\romannumeral\bbl@tempa>\{the\toks@\}}%

```

```

3120      \\bbl@tglobal\<extras\language>}%
3121      \fi
3122      \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.

```

3123 \def\bbl@chapttype{chapter}
3124 \ifx\@makechapterhead\@undefined
3125   \let\bbl@patchchapter\relax
3126 \else\ifx\thechapter\@undefined
3127   \let\bbl@patchchapter\relax
3128 \else\ifx\ps@headings\@undefined
3129   \let\bbl@patchchapter\relax
3130 \else
3131   \def\bbl@patchchapter{%
3132     \global\let\bbl@patchchapter\relax
3133     \gdef\bbl@chfmt{%
3134       \bbl@ifunset{bbl@\bbl@chapttype fmt@\language}%
3135       {\@chapapp\space\thechapter}
3136       {\@nameuse{bbl@\bbl@chapttype fmt@\language}}}
3137     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3138     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3139     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3140     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3141     \bbl@tglobal\appendix
3142     \bbl@tglobal\ps@headings
3143     \bbl@tglobal\chaptermark
3144     \bbl@tglobal\@makechapterhead}
3145   \let\bbl@patchappendix\bbl@patchchapter
3146 \fi\fi\fi
3147 \ifx\@part\@undefined
3148   \let\bbl@patchpart\relax
3149 \else
3150   \def\bbl@patchpart{%
3151     \global\let\bbl@patchpart\relax
3152     \gdef\bbl@partformat{%
3153       \bbl@ifunset{bbl@partfmt@\language}%
3154       {\partname\nobreakspace\thepart}
3155       {\@nameuse{bbl@partfmt@\language}}}
3156     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3157     \bbl@tglobal\@part}
3158 \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

```

3159 \let\bbl@calendar\@empty
3160 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3161 \def\bbl@localedate#1#2#3#4{%
3162   \begingroup
3163     \edef\bbl@they{#2}%
3164     \edef\bbl@them{#3}%
3165     \edef\bbl@thed{#4}%
3166     \edef\bbl@tempe{%
3167       \bbl@ifunset{bbl@calpr@\language}{\bbl@cl{calpr}},%
3168       #1}%
3169     \bbl@replace\bbl@tempe{ }{}%
3170     \bbl@replace\bbl@tempe{CONVERT}{convert}% Hackish
3171     \bbl@replace\bbl@tempe{convert}{convert}%
3172     \let\bbl@ld@calendar\@empty
3173     \let\bbl@ld@variant\@empty
3174     \let\bbl@ld@convert\relax
3175     \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld@##1}{##2}}%

```

```

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

```

**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).

```

3224 \let\bbl@calendar\empty
3225 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3226 \@nameuse{\bbl@ca@#2}#1\@}
3227 \newcommand\BabelDateSpace{\nobreakspace}
3228 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3229 \newcommand\BabelDated[1][{\number#1}]
3230 \newcommand\BabelDatedd[1][{\ifnum#1<10 0\fi\number#1}]
3231 \newcommand\BabelDateM[1][{\number#1}]
3232 \newcommand\BabelDateMM[1][{\ifnum#1<10 0\fi\number#1}]

```

```

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

```

#### Transforms.

```

3267 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3268 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3269 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3270 #1[#2]{#3}{#4}{#5}}
3271 \begingroup % A hack. TODO. Don't require an specific order
3272 \catcode`\%=12
3273 \catcode`\&=14
3274 \gdef\bbl@transforms#1#2#3{%&
3275 \directlua{
3276 local str = [==[#2]==]
3277 str = str:gsub('%.%d+%.%d+$', '')
3278 token.set_macro('babeltempa', str)
3279 }&
3280 \def\babeltempc{}&
3281 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3282 \ifin@ \else
3283 \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&
3284 \fi
3285 \ifin@
3286 \bbl@foreach\bbl@KVP@transforms{%&
3287 \bbl@xin@{: \babeltempa,}{,##1,}&
3288 \ifin@ &% font:font:transform syntax
3289 \directlua{
3290 local t = {}
3291 for m in string.gmatch('##1'..'':, '(-.):') do
3292 table.insert(t, m)
3293 end

```



```

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

```

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

```

3321 \def\bbl@provide@lsys#1{%
3322     \bbl@ifunset{bbl@lname@#1}%
3323     {\bbl@load@info{#1}}%
3324     }%
3325     \bbl@csarg\let{lsys@#1}\@empty
3326     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3327     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3328     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3329     \bbl@ifunset{bbl@lname@#1}{}%
3330     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3331     \ifcase\bbl@engine\or\or
3332         \bbl@ifunset{bbl@prehc@#1}{}%
3333         {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3334          }%
3335         {\ifx\bbl@xenohyph\undefined
3336          \global\let\bbl@xenohyph\bbl@xenohyph@d
3337          \ifx\AtBeginDocument\@notprerr
3338              \expandafter\@secondoftwo % to execute right now
3339          \fi
3340          \AtBeginDocument{%
3341              \bbl@patchfont{\bbl@xenohyph}%
3342              {\expandafter\select@language\expandafter{\language}}}%
3343          \fi}}%
3344     \fi
3345     \bbl@csarg\bbl@tglobal{lsys@#1}}
3346 \def\bbl@xenohyph@d{%
3347     \bbl@ifset{bbl@prehc@language}%
3348     {\ifnum\hyphenchar\font=\defaultshyphenchar
3349      \iffontchar\font\bbl@cl{prehc}\relax
3350      \hyphenchar\font\bbl@cl{prehc}\relax
3351      \else\iffontchar\font"200B
3352      \hyphenchar\font"200B
3353      \else

```

```

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

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3366 \def\bbl@load@info#1{%
3367 \def\BabelBeforeIni##1##2{%
3368 \begingroup
3369 \bbl@read@ini{##1}0%
3370 \endinput % babel- .tex may contain onlypreamble's
3371 \endgroup}% boxed, to avoid extra spaces:
3372 {\bbl@input@texini{#1}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T<sub>E</sub>X. Non-digits characters are kept. The first macro is the generic “localized” command.

```

3373 \def\bbl@setdigits#1#2#3#4#5{%
3374 \bbl@exp{%
3375 \def<\language name digits>####1{% ie, \langdigits
3376 \<\bbl@digits@\language name>####1\\\@nil}%
3377 \let<\bbl@cntr@digits@\language name>\<\language name digits>%
3378 \def<\language name counter>####1{% ie, \langcounter
3379 \\\expandafter\<\bbl@counter@\language name>%
3380 \\\csname c@####1\endcsname}%
3381 \def<\bbl@counter@\language name>####1{% ie, \bbl@counter@lang
3382 \\\expandafter\<\bbl@digits@\language name>%
3383 \\\number####1\\\@nil}}%
3384 \def\bbl@tempa##1##2##3##4##5{%
3385 \bbl@exp{% Wow, quite a lot of hashes! :- (
3386 \def<\bbl@digits@\language name>#####1{%
3387 \\\ifx#####1\\\@nil % ie, \bbl@digits@lang
3388 \\\else
3389 \\\ifx0#####1#1%
3390 \\\else\\\ifx1#####1#2%
3391 \\\else\\\ifx2#####1#3%
3392 \\\else\\\ifx3#####1#4%
3393 \\\else\\\ifx4#####1#5%
3394 \\\else\\\ifx5#####1#1%
3395 \\\else\\\ifx6#####1#2%
3396 \\\else\\\ifx7#####1#3%
3397 \\\else\\\ifx8#####1#4%
3398 \\\else\\\ifx9#####1#5%
3399 \\\else#####1%
3400 \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3401 \\\expandafter\<\bbl@digits@\language name>%
3402 \\\fi}}}%
3403 \bbl@tempa}

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3404 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3405 \ifx\#1% % \ before, in case #1 is multiletter
3406 \bbl@exp{%

```

```

3407 \def\\bbl@tempa####1{%
3408 \<ifcase>####1\space\the\toks@<else>\\@ctrerr<fi>}}%
3409 \else
3410 \toks@<expandafter{\the\toks@<or #1>%
3411 \expandafter\bbl@buildifcase
3412 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collect 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).

```

3413 \newcommand\localenumberal[2]{\bbl@cs{cnt@#1@<language>}{#2}}
3414 \def\bbl@localecnt#1#2{\localenumberal{#2}{#1}}
3415 \newcommand\localecounter[2]{%
3416 \expandafter\bbl@localecnt
3417 \expandafter{\number\csname c@#2\endcsname}{#1}}
3418 \def\bbl@alphnumeral#1#2{%
3419 \expandafter\bbl@alphnumeral@i\number#2 76543210\\@{#1}}
3420 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\\@#9{%
3421 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3422 \bbl@alphnumeral@ii{#9}000000#1\or
3423 \bbl@alphnumeral@ii{#9}00000#1#2\or
3424 \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3425 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3426 \bbl@alphnum@invalid{>9999}%
3427 \fi}
3428 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3429 \bbl@ifunset{bbl@cnt@#1.F.\number#5#6#7#8@<language>}%
3430 {\bbl@cs{cnt@#1.4@<language>}{#5}
3431 \bbl@cs{cnt@#1.3@<language>}{#6}
3432 \bbl@cs{cnt@#1.2@<language>}{#7}
3433 \bbl@cs{cnt@#1.1@<language>}{#8}
3434 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3435 \bbl@ifunset{bbl@cnt@#1.S.321@<language>}{}%
3436 {\bbl@cs{cnt@#1.S.321@<language>}}%
3437 \fi}%
3438 {\bbl@cs{cnt@#1.F.\number#5#6#7#8@<language>}}
3439 \def\bbl@alphnum@invalid#1{%
3440 \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.

```

3441 \def\bbl@localeinfo#1#2{%
3442 \bbl@ifunset{bbl@info@#2}{#1}%
3443 {\bbl@ifunset{bbl@<csname> bbl@info@#2\endcsname @<language>}{#1}%
3444 {\bbl@cs{<csname> bbl@info@#2\endcsname @<language>}}}
3445 \newcommand\localeinfo[1]{%
3446 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3447 \bbl@afterelse\bbl@localeinfo}%
3448 \else
3449 \bbl@localeinfo
3450 {\bbl@error{no-ini-info}{}}}%
3451 {#1}%
3452 \fi}
3453 % \namedef{bbl@info@<name>.<locale>}{<lname>}
3454 \namedef{bbl@info@tag.ini}{<lini>}
3455 \namedef{bbl@info@<name>.english}{<elname>}
3456 \namedef{bbl@info@<name>.opentype}{<lname>}
3457 \namedef{bbl@info@tag.bcp47}{<tbcp>}
3458 \namedef{bbl@info@<language>.tag.bcp47}{<lbcp>}
3459 \namedef{bbl@info@tag.opentype}{<lotf>}
3460 \namedef{bbl@info@script.<name>}{<esname>}
3461 \namedef{bbl@info@script.<name>.opentype}{<sname>}

```

```

3462 \@namedef{bbl@info@script.tag.bcp47}{sbcpr}
3463 \@namedef{bbl@info@script.tag.opentype}{sotfr}
3464 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3465 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3466 \@namedef{bbl@info@extension.t.tag.bcp47}{exttr}
3467 \@namedef{bbl@info@extension.u.tag.bcp47}{extur}
3468 \@namedef{bbl@info@extension.x.tag.bcp47}{extxr}

```

TeX 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.

```

3469 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3470   \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3471 \else
3472   \def\bbl@uftocode#1{\expandafter\string#1}
3473 \fi
3474 % Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
3475 % expandable (|\bbl@ifsamestring| isn't).
3476 \providecommand\BCPdata{}
3477 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3478   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3479   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3480     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3481     {\bbl@bcpdata@ii#6}\bbl@main@language}%
3482     {\bbl@bcpdata@ii#1#2#3#4#5#6}\language}%
3483   \def\bbl@bcpdata@ii#1#2{%
3484     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3485     {\bbl@error{unknown-ini-field}{#1}{}}%
3486     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3487     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3488 \fi
3489 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3490 \newcommand\BabelUppercaseMapping[3]{%
3491   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3492 \newcommand\BabelTitlecaseMapping[3]{%
3493   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3494 \newcommand\BabelLowercaseMapping[3]{%
3495   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

```

The parser for casing and casing.⟨variant⟩.

```

3496 \def\bbl@casemapping#1#2#3{% 1:variant
3497   \def\bbl@tempa##1 ##2{% Loop
3498     \bbl@casemapping@i{##1}%
3499     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3500   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3501   \def\bbl@tempe{0}% Mode (upper/lower...)
3502   \def\bbl@tempc{#3}% Casing list
3503   \expandafter\bbl@tempa\bbl@tempc\@empty}
3504 \def\bbl@casemapping@i#1{%
3505   \def\bbl@tempb{#1}%
3506   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3507     \@nameuse{regex_replace_all:nnN}%
3508     {[\\x{c0}-\\x{ff}][\\x{80}-\\x{bf}]*}{\\0}}\bbl@tempb
3509   \else
3510     \@nameuse{regex_replace_all:nnN}{.}{\\0}}\bbl@tempb % TODO. needed?
3511   \fi
3512   \expandafter\bbl@casemapping@ii\bbl@tempb\@}
3513 \def\bbl@casemapping@ii#1#2#3\@{%
3514   \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3515   \ifin@
3516     \edef\bbl@tempe{%
3517       \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3518   \else
3519     \ifcase\bbl@tempe\relax

```

```

3520 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3521 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3522 \or
3523 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3524 \or
3525 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3526 \or
3527 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3528 \fi
3529 \fi}

```

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

```

3530 <<(*More package options)>> ≡
3531 \DeclareOption{ensureinfo=off}{}
3532 <</More package options>>
3533 \let\bbl@ensureinfo\@gobble
3534 \newcommand\BabelEnsureInfo{%
3535   \ifx\InputIfFileExists\@undefined\else
3536     \def\bbl@ensureinfo##1{%
3537       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3538   \fi
3539   \bbl@foreach\bbl@loaded{{%
3540     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3541     \def\language{##1}%
3542     \bbl@ensureinfo{##1}}}
3543   \ifpackage{babel}{ensureinfo=off}}%
3544   {\AtEndOfPackage{% Test for plain.
3545     \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3546 \newcommand\getlocaleproperty{%
3547   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3548 \def\bbl@getproperty@s#1#2#3{%
3549   \let#1\relax
3550   \def\bbl@elt##1##2##3{%
3551     \bbl@ifsamestring{##1/##2}{#3}%
3552     {\providecommand#1{##3}%
3553     \def\bbl@elt###1####2####3{}}}%
3554   {}}%
3555   \bbl@cs{inidata@#2}}%
3556 \def\bbl@getproperty@x#1#2#3{%
3557   \bbl@getproperty@s{#1}{#2}{#3}%
3558   \ifx#1\relax
3559     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3560   \fi}
3561 \let\bbl@ini@loaded\@empty
3562 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3563 \def\ShowLocaleProperties#1{%
3564   \typeout{}}%
3565   \typeout{*** Properties for language '#1' ***}
3566   \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3567   \@nameuse{bbl@inidata@#1}%
3568   \typeout{*****}}

```

## 5 Adjusting the Babel behavior

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

```

3569 \newcommand\babeladjust[1]{% TODO. Error handling.
3570   \bbl@forkv{#1}{%
3571     \bbl@ifunset{bbl@ADJ@##1@##2}%

```

```

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

```

```

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

```

3686 <<*More package options>> ≡
3687 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3688 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3689 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3690 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3691 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3692 <</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.

```

3693 \bbl@trace{Cross referencing macros}
3694 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3695   \def\@newl@bel#1#2#3{%
3696     {\@safe@activestrue
3697       \bbl@ifunset{#1#2}%
3698       \relax
3699       {\gdef\@multiplelabels{%
3700         \@latex@warning@no@line{There were multiply-defined labels}}%
3701         \@latex@warning@no@line{Label `#2' multiply defined}}}%
3702     \global\@namedef{#1#2}{#3}}

```

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

```

3703 \CheckCommand*\@testdef[3]{%
3704   \def\reserved@a{#3}%
3705   \expandafter\ifx\csname#1#2\endcsname\reserved@a
3706   \else
3707     \@tempswatrue
3708   \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.

```

3709 \def\@testdef#1#2#3{% TODO. With @samestring?
3710   \@safe@activestrue
3711   \expandafter\let\expandafter\bbl@tempa\csname #1#2\endcsname
3712   \def\bbl@tempb{#3}%
3713   \@safe@activesfalse
3714   \ifx\bbl@tempa\relax
3715   \else
3716     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3717   \fi
3718   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3719   \ifx\bbl@tempa\bbl@tempb
3720   \else
3721     \@tempswatrue
3722   \fi}
3723 \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.

```

3724 \bbl@xin@{R}\bbl@opt@safe
3725 \ifin@
3726   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3727   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3728   {\expandafter\strip@prefix\meaning\ref}%
3729   \ifin@

```



```

3730 \bbl@redefine\@kernel@ref#1{%
3731 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activfalse}
3732 \bbl@redefine\@kernel@pageref#1{%
3733 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3734 \bbl@redefine\@kernel@sref#1{%
3735 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3736 \bbl@redefine\@kernel@spageref#1{%
3737 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3738 \else
3739 \bbl@redefineroast\ref#1{%
3740 \@safe@activetrue\org@@ref{#1}\@safe@activfalse}
3741 \bbl@redefineroast\pageref#1{%
3742 \@safe@activetrue\org@@pageref{#1}\@safe@activfalse}
3743 \fi
3744 \else
3745 \let\org@ref\ref
3746 \let\org@pageref\pageref
3747 \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.

```

3748 \bbl@xin@{B}\bbl@opt@safe
3749 \ifin@
3750 \bbl@redefine\@citex[#1]#2{%
3751 \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activfalse
3752 \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.

```

3753 \AtBeginDocument{%
3754 \@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.)

```

3755 \def\@citex[#1][#2]#3{%
3756 \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activfalse
3757 \org@@citex[#1][#2]{\bbl@tempa}}%
3758 }{}

```

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

```

3759 \AtBeginDocument{%
3760 \@ifpackageloaded{cite}{%
3761 \def\@citex[#1]#2{%
3762 \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3763 }{}

```

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

```

3764 \bbl@redefine\nocite#1{%
3765 \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

`\bibcite` The macro that is used in the `.aux` file to define citation labels. When packages such as `natbib` or `cite` are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where `\@safe@activetrue` is in effect. This switch needs to be reset inside the `\hbox` which contains the citation label. In order to determine during `.aux` file processing which definition of `\bibcite` is needed we define `\bibcite`

in such a way that it redefines itself with the proper definition. We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
3766 \bbl@redefine\bibcite{%
3767   \bbl@cite@choice
3768   \bibcite}
```

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

```
3769 \def\bbl@bibcite#1#2{%
3770   \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.

```
3771 \def\bbl@cite@choice{%
3772   \global\let\bibcite\bbl@bibcite
3773   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3774   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3775   \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.

```
3776 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3777 \bbl@redefine\@bibitem#1{%
3778   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3779 \else
3780   \let\org@nocite\nocite
3781   \let\org@citex\citex
3782   \let\org@bibcite\bibcite
3783   \let\org@bibitem\@bibitem
3784 \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.

```
3785 \bbl@trace{Marks}
3786 \IfBabelLayout{sectioning}
3787   {\ifx\bbl@opt@headfoot\@nnil
3788     \g@addto@macro\@resetactivechars{%
3789       \set@typeset@protect
3790       \expandafter\select@language\expandafter{\bbl@main@language}%
3791       \let\protect\noexpand
3792       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3793         \edef\thepage{%
3794           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3795       \fi}%
3796   \fi}
3797 {\ifbbl@single\else
3798   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3799     \markright#1{%
3800       \bbl@ifblank{#1}%
3801       {\org@markright{}}}%
3802     {\toks@{#1}%
3803       \bbl@exp{%
3804         \\org@markright{\\protect\\foreignlanguage{\\language}%
3805           {\\protect\\bbl@restore@actives\\the\\toks@}}}%
3806       }%
3807     }%
3808   }
```

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

```

3806 \ifx\@mkboth\markboth
3807 \def\bbl@tempc{\let\@mkboth\markboth}%
3808 \else
3809 \def\bbl@tempc{%
3810 \fi
3811 \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3812 \markboth#1#2{%
3813 \protected@edef\bbl@tempb##1{%
3814 \protect\foreignlanguage
3815 {\language\name}{\protect\bbl@restore@actives##1}}%
3816 \bbl@ifblank{#1}%
3817 {\toks@{}}%
3818 {\toks@\expandafter{\bbl@tempb{#1}}}%
3819 \bbl@ifblank{#2}%
3820 {\@temptokena{}}%
3821 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3822 \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}}%
3823 \bbl@tempc
3824 \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.

```

3825 \bbl@trace{Preventing clashes with other packages}
3826 \ifx\org@ref\undefined\else
3827 \bbl@xin@{R}\bbl@opt@safe
3828 \ifin@
3829 \AtBeginDocument{%
3830 \ifpackageloaded{ifthen}{%
3831 \bbl@redefine@long\ifthenelse#1#2#3{%
3832 \let\bbl@temp@pref\pageref
3833 \let\pageref\org@pageref
3834 \let\bbl@temp@ref\ref
3835 \let\ref\org@ref
3836 \@safe@activestru
3837 \org@ifthenelse{#1}%
3838 {\let\pageref\bbl@temp@pref
3839 \let\ref\bbl@temp@ref
3840 \@safe@activesfalse
3841 #2}%

```

```

3842         {\let\pageref\bbbl@temp@pref
3843         \let\ref\bbbl@temp@ref
3844         \@safe@activesfalse
3845         #3}%
3846     }%
3847 }{}%
3848 }
3849 \fi

```

### 5.3.2 varioref

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

```

3850 \AtBeginDocument{%
3851     \@ifpackageloaded{varioref}{%
3852         \bbbl@redefine\@vppageref#1[#2]#3{%
3853             \@safe@activestrue
3854             \org@@@vppageref{#1}[#2]#3}%
3855         \@safe@activesfalse}%
3856     \bbbl@redefine\vrefpagemum#1#2{%
3857         \@safe@activestrue
3858         \org@vrefpagemum{#1}#2}%
3859     \@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.

```

3860     \expandafter\def\csname Ref \endcsname#1{%
3861         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3862     }{}%
3863 }
3864 \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.

```

3865 \AtEndOfPackage{%
3866     \AtBeginDocument{%
3867         \@ifpackageloaded{hhline}%
3868         {\expandafter\ifx\csname normal@char\string\endcsname\relax
3869             \else
3870                 \makeatletter
3871                 \def\@currname{hhline}\input{hhline.sty}\makeatother
3872             \fi}%
3873         {}}}

```

`\substitutefontfamily` *Deprecated.* Use the tools provided by L<sup>A</sup>T<sub>E</sub>X. 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.

```

3874 \def\substitutefontfamily#1#2#3{%
3875     \lowercase{\immediate\openout15=#1#2.fd\relax}%
3876     \immediate\write15{%
3877         \string\ProvidesFile{#1#2.fd}%
3878         [\the\year/\two@digits{\the\month}/\two@digits{\the\day}]
3879         \space generated font description file]^^J

```

```

3880 \string\DeclareFontFamily{#1}{#2}{}}^^J
3881 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^^J
3882 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^^J
3883 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^^J
3884 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^^J
3885 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^^J
3886 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^^J
3887 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^^J
3888 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^^J
3889 }%
3890 \closeout15
3891 }
3892 \@onlypreamble\substitutefontfamily

```

## 5.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  and  $\mathrm{L}_{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$  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

3893 \bbl@trace{Encoding and fonts}
3894 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3895 \newcommand\BabelNonText{TS1,T3,TS3}
3896 \let\org@TeX\TeX
3897 \let\org@LaTeX\LaTeX
3898 \let\ensureascii@firstofone
3899 \let\asciientcoding\empty
3900 \AtBeginDocument{%
3901   \def\@elt#1{,#1,}%
3902   \edef\bbl@tempa{\expandafter\@gobbletwo\fontenc@load@list}%
3903   \let\@elt\relax
3904   \let\bbl@tempb\empty
3905   \def\bbl@tempc{OT1}%
3906   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3907     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3908   \bbl@foreach\bbl@tempa{%
3909     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3910     \ifin@
3911       \def\bbl@tempb{#1}% Store last non-ascii
3912     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3913     \ifin@else
3914       \def\bbl@tempc{#1}% Store last ascii
3915     \fi
3916   \fi}%
3917   \ifx\bbl@tempb\empty\else
3918     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3919     \ifin@else
3920       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3921     \fi
3922     \let\asciientcoding\bbl@tempc
3923     \renewcommand\ensureascii[1]{%
3924       {\fontencoding{\asciientcoding}\selectfont#1}}%
3925     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3926     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3927   \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.

```
3928 \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.

```
3929 \AtBeginDocument{%
3930   \ifpackageloaded{fontspec}%
3931     {\xdef\latinencoding{%
3932       \ifx\UTFencname\undefined
3933         EU\ifcase\bb@engine\or2\or1\fi
3934       \else
3935         \UTFencname
3936       \fi}}%
3937   {\gdef\latinencoding{OT1}%
3938     \ifx\cf@encoding\bb@t@one
3939       \xdef\latinencoding{\bb@t@one}%
3940     \else
3941       \def\@elt#1{,#1,}%
3942       \edef\bb@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3943       \let\@elt\relax
3944       \bb@xin@{,T1,}\bb@tempa
3945       \ifin@
3946         \xdef\latinencoding{\bb@t@one}%
3947       \fi
3948     \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.

```
3949 \DeclareRobustCommand{\latintext}{%
3950   \fontencoding{\latinencoding}\selectfont
3951   \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.

```
3952 \ifx\@undefined\DeclareTextFontCommand
3953   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3954 \else
3955   \DeclareTextFontCommand{\textlatin}{\latintext}
3956 \fi
```

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

```
3957 \def\bb@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 T<sub>E</sub>X 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 LuaT<sub>E</sub>X-j<sub>a</sub> shows, vertical typesetting is possible, too.

```

3958 \bbl@trace{Loading basic (internal) bidi support}
3959 \ifodd\bbl@engine
3960 \else % TODO. Move to txtbabel
3961   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3962     \bbl@error{bidi-only-lua}{}}{}%
3963     \let\bbl@beforeforeign\leavevmode
3964     \AtEndOfPackage{%
3965       \EnableBabelHook{babel-bidi}%
3966       \bbl@xebidipar}
3967   \fi\fi
3968   \def\bbl@loadxebidi#1{%
3969     \ifx\RTLfootnotetext\@undefined
3970       \AtEndOfPackage{%
3971         \EnableBabelHook{babel-bidi}%
3972         \bbl@loadfontspec % bidi needs fontspec
3973         \usepackage#1{bidi}%
3974         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3975         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3976           \ifnum\@nameuse{bbl@wdir@\language}\tw@ % 'AL' bidi
3977             \bbl@digitsdotdash % So ignore in 'R' bidi
3978           \fi}}%
3979     \fi}
3980   \ifnum\bbl@bidimode>200 % Any xe bidi=
3981     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3982       \bbl@tentative{bidi=bidi}
3983       \bbl@loadxebidi{}
3984     \or
3985       \bbl@loadxebidi{[rldocument]}
3986     \or
3987       \bbl@loadxebidi{}
3988     \fi
3989   \fi
3990 \fi
3991 % TODO? Separate:
3992 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
3993   \let\bbl@beforeforeign\leavevmode
3994   \ifodd\bbl@engine
3995     \newattribute\bbl@attr@dir
3996     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3997     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3998   \fi
3999   \AtEndOfPackage{%
4000     \EnableBabelHook{babel-bidi}%
4001     \ifodd\bbl@engine\else
4002       \bbl@xebidipar
4003     \fi}
4004 \fi

```

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

```

4005 \bbl@trace{Macros to switch the text direction}
4006 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4007 \def\bbl@rscripts{% TODO. Base on codes ??
4008   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4009   Old Hungarian,Lydian,Mandaean,Manichaean,%
4010   Meroitic Cursive,Meroitic,Old North Arabian,%

```

```

4011 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4012 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4013 Old South Arabian,}%
4014 \def\bbbl@provide@dirs#1{%
4015   \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
4016   \ifin@
4017     \global\bbbl@csarg\chardef{wdir@#1}\@ne
4018     \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
4019     \ifin@
4020       \global\bbbl@csarg\chardef{wdir@#1}\tw@
4021       \fi
4022   \else
4023     \global\bbbl@csarg\chardef{wdir@#1}\z@
4024   \fi
4025   \ifodd\bbbl@engine
4026     \bbbl@csarg\ifcase{wdir@#1}%
4027       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
4028     \or
4029       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
4030     \or
4031       \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
4032     \fi
4033   \fi}
4034 \def\bbbl@switchdir{%
4035   \bbbl@ifunset{bbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}{}%
4036   \bbbl@ifunset{bbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}{}%
4037   \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}
4038 \def\bbbl@setdirs#1{% TODO - math
4039   \ifcase\bbbl@select@type % TODO - strictly, not the right test
4040     \bbbl@bodydir{#1}%
4041     \bbbl@pardir{#1}% <- Must precede \bbbl@texmdir
4042   \fi
4043   \bbbl@texmdir{#1}}
4044 % TODO. Only if \bbbl@bidimode > 0?:
4045 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4046 \DisableBabelHook{babel-bidi}

```

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

```

4047 \ifodd\bbbl@engine % luatex=1
4048 \else % pdftex=0, xetex=2
4049   \newcount\bbbl@dirlevel
4050   \chardef\bbbl@thetexmdir\z@
4051   \chardef\bbbl@thepardir\z@
4052   \def\bbbl@texmdir#1{%
4053     \ifcase#1\relax
4054       \chardef\bbbl@thetexmdir\z@
4055       \@nameuse{setlatin}%
4056       \bbbl@texmdir@i\beginL\endL
4057     \else
4058       \chardef\bbbl@thetexmdir\@ne
4059       \@nameuse{setnonlatin}%
4060       \bbbl@texmdir@i\beginR\endR
4061     \fi}
4062   \def\bbbl@texmdir@i#1#2{%
4063     \ifhmode
4064       \ifnum\currentgrouplevel>\z@
4065         \ifnum\currentgrouplevel=\bbbl@dirlevel
4066           \bbbl@error{multiple-bidi}{}{}%
4067           \bgroup\aftergroup#2\aftergroup\egroup
4068         \else
4069           \ifcase\currentgrouptype\or % 0 bottom
4070             \aftergroup#2% 1 simple {}
4071           \or

```



```

4072      \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4073      \or
4074      \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4075      \or\or\or % vbox vtop align
4076      \or
4077      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4078      \or\or\or\or\or\or % output math disc insert vcent mathchoice
4079      \or
4080      \aftergroup#2% 14 \begingroup
4081      \else
4082      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4083      \fi
4084      \fi
4085      \bbl@dirlevel\currentgrouplevel
4086      \fi
4087      #1%
4088      \fi}
4089      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4090      \let\bbl@bodydir\@gobble
4091      \let\bbl@pagedir\@gobble
4092      \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).

```

4093      \def\bbl@xebidipar{%
4094      \let\bbl@xebidipar\relax
4095      \TeXeTstate\@ne
4096      \def\bbl@xeverypar{%
4097      \ifcase\bbl@thepardir
4098      \ifcase\bbl@thetextdir\else\beginR\fi
4099      \else
4100      {\setbox\z@\lastbox\beginR\box\z@}%
4101      \fi}%
4102      \let\bbl@severypar\everypar
4103      \newtoks\everypar
4104      \everypar=\bbl@severypar
4105      \bbl@severypar{\bbl@xeverypar\the\everypar}}
4106      \ifnum\bbl@bidimode>200 % Any xe bidi=
4107      \let\bbl@textdir\i\@gobbletwo
4108      \let\bbl@xebidipar\@empty
4109      \AddBabelHook{bidi}{foreign}{%
4110      \def\bbl@tempa{\def\BabelText###1}%
4111      \ifcase\bbl@thetextdir
4112      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4113      \else
4114      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4115      \fi}
4116      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4117      \fi
4118      \fi

```

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

```

4119      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4120      \AtBeginDocument{%
4121      \ifx\pdfstringdefDisableCommands\undefined\else
4122      \ifx\pdfstringdefDisableCommands\relax\else
4123      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4124      \fi
4125      \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`.

```
4126 \bbl@trace{Local Language Configuration}
4127 \ifx\loadlocalcfg\undefined
4128   \@ifpackagewith{babel}{noconfigs}%
4129     {\let\loadlocalcfg@gobble}%
4130     {\def\loadlocalcfg#1{%
4131       \InputIfFileExists{#1.cfg}%
4132       {\typeout{*****^J%
4133                * Local config file #1.cfg used^^J%
4134                *}}}%
4135       \@empty}}
4136 \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).

```
4137 \bbl@trace{Language options}
4138 \let\bbl@afterlang\relax
4139 \let\BabelModifiers\relax
4140 \let\bbl@loaded\@empty
4141 \def\bbl@load@language#1{%
4142   \InputIfFileExists{#1.ldf}%
4143   {\edef\bbl@loaded{\CurrentOption
4144     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4145     \expandafter\let\expandafter\bbl@afterlang
4146     \csname\CurrentOption.ldf-h@k\endcsname
4147     \expandafter\let\expandafter\BabelModifiers
4148     \csname bbl@mod@\CurrentOption\endcsname
4149     \bbl@exp{\AtBeginDocument{%
4150       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4151     {\IfFileExists{babel-#1.tex}%
4152      {\def\bbl@tempa{%
4153        .\There is a locale ini file for this language.\\
4154        If it's the main language, try adding `provide='\\
4155        to the babel package options}}%
4156      {\let\bbl@tempa\empty}%
4157      \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.

```
4158 \def\bbl@try@load@lang#1#2#3{%
4159   \IfFileExists{\CurrentOption.ldf}%
4160   {\bbl@load@language{\CurrentOption}}%
4161   {\bbl@load@language{#2#3}}
4162 %
4163 \DeclareOption{hebrew}{%
4164   \ifcase\bbl@engine\or
4165     \bbl@error{only-pdftex-lang}{hebrew}{luatex}}%
4166 \fi
4167 \input{rlbabel.def}%
4168 \bbl@load@language{hebrew}}
4169 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}}
4170 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}}
4171 \DeclareOption{polutonikogreek}{%
```

```

4172 \bbl@try@load@lang{{greek}}{\languageattribute{greek}{polutoniko}}
4173 \DeclareOption{russian}{\bbl@try@load@lang{{russianb}}{}}
4174 \DeclareOption{ukrainian}{\bbl@try@load@lang{{ukraineb}}{}}
4175 \DeclareOption{uppertsorbian}{\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.

```

4176 \ifx\bbl@opt@config@nnil
4177 \@ifpackagewith{babel}{noconfigs}{}%
4178 {\InputIfFileExists{bblopts.cfg}%
4179 {\typeout{*****^J%
4180          * Local config file bblopts.cfg used^^J%
4181          *}}%
4182 {}}%
4183 \else
4184 \InputIfFileExists{\bbl@opt@config.cfg}%
4185 {\typeout{*****^J%
4186          * Local config file \bbl@opt@config.cfg used^^J%
4187          *}}%
4188 {\bbl@error{config-not-found}{}}}%
4189 \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.

```

4190 \ifx\bbl@opt@main@nnil
4191 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4192 \let\bbl@tempb\@empty
4193 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4194 \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4195 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4196 \ifx\bbl@opt@main@nnil % ie, if not yet assigned
4197 \ifodd\bbl@iniflag % = *=
4198 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4199 \else % n +=
4200 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}}%
4201 \fi
4202 \fi}%
4203 \fi
4204 \else
4205 \bbl@info{Main language set with 'main='. Except if you have\\%
4206          problems, prefer the default mechanism for setting\\%
4207          the main language, ie, as the last declared.\\%
4208          Reported}
4209 \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`).

```

4210 \ifx\bbl@opt@main@nnil\else
4211 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4212 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4213 \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.

```

4214 \bbl@foreach\bbl@language@opts{%
4215 \def\bbl@tempa{#1}%
4216 \ifx\bbl@tempa\bbl@opt@main\else

```

```

4217 \ifnum\bb@iniflag<\tw@ % 0 0 (other = ldf)
4218 \bb@ifunset{ds@#1}%
4219 {\DeclareOption{#1}{\bb@load@language{#1}}}%
4220 {}%
4221 \else % + * (other = ini)
4222 \DeclareOption{#1}{%
4223 \bb@ldfinit
4224 \babelprovide[import]{#1}%
4225 \bb@afterldf{}}%
4226 \fi
4227 \fi}
4228 \bb@foreach\@classoptionslist{%
4229 \def\bb@tempa{#1}%
4230 \ifx\bb@tempa\bb@opt@main\else
4231 \ifnum\bb@iniflag<\tw@ % 0 0 (other = ldf)
4232 \bb@ifunset{ds@#1}%
4233 {\IfFileExists{#1.ldf}%
4234 {\DeclareOption{#1}{\bb@load@language{#1}}}%
4235 {}}%
4236 {}%
4237 \else % + * (other = ini)
4238 \IfFileExists{babel-#1.tex}%
4239 {\DeclareOption{#1}{%
4240 \bb@ldfinit
4241 \babelprovide[import]{#1}%
4242 \bb@afterldf{}}}%
4243 {}%
4244 \fi
4245 \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):

```

4246 \def\AfterBabelLanguage#1{%
4247 \bb@ifsamestring\CurrentOption{#1}{\global\bb@add\bb@afterlang{}}
4248 \DeclareOption*{}
4249 \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.

```

4250 \bb@trace{Option 'main'}
4251 \ifx\bb@opt@main\@nnil
4252 \edef\bb@tempa{\@classoptionslist,\bb@language@opts}
4253 \let\bb@tempc\@empty
4254 \edef\bb@templ{,\bb@loaded,}
4255 \edef\bb@templ{\expandafter\strip@prefix\meaning\bb@templ}
4256 \bb@for\bb@tempb\bb@tempa{%
4257 \edef\bb@tempd{,\bb@tempb,}%
4258 \edef\bb@tempd{\expandafter\strip@prefix\meaning\bb@tempd}%
4259 \bb@xin@{\bb@tempd}{\bb@templ}%
4260 \ifin@\edef\bb@tempc{\bb@tempb}\fi}
4261 \def\bb@tempa#1,#2\@nnil{\def\bb@tempb{#1}}
4262 \expandafter\bb@tempa\bb@loaded,\@nnil
4263 \ifx\bb@tempb\bb@tempc\else
4264 \bb@warning{%
4265 Last declared language option is '\bb@tempc',\%
4266 but the last processed one was '\bb@tempb'.\%
4267 The main language can't be set as both a global\%
4268 and a package option. Use 'main=\bb@tempc' as\%

```

```

4269     option. Reported}
4270 \fi
4271 \else
4272 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4273 \bbl@ldfinit
4274 \let\CurrentOption\bbl@opt@main
4275 \bbl@exp{% \bbl@opt@provide = empty if *
4276     \\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4277 \bbl@afterldf{}
4278 \DeclareOption{\bbl@opt@main}{}
4279 \else % case 0,2 (main is ldf)
4280 \ifx\bbl@loadmain\relax
4281 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4282 \else
4283 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4284 \fi
4285 \ExecuteOptions{\bbl@opt@main}
4286 \@namedef{ds@\bbl@opt@main}{}%
4287 \fi
4288 \DeclareOption*{}
4289 \ProcessOptions*
4290 \fi
4291 \bbl@exp{%
4292     \\AtBeginDocument{\\bbl@usehooks@lang{/}{\begindocument}{}}}%
4293 \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.

```

4294 \ifx\bbl@main@language\undefined
4295 \bbl@info{%
4296     You haven't specified a language as a class or package\\%
4297     option. I'll load 'nil'. Reported}
4298 \bbl@load@language{nil}
4299 \fi
4300 \</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  $\TeX$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\TeX$  can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain  $\TeX$  and  $\LaTeX$ , some of it is for the  $\LaTeX$  case only.

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

A proxy file for `switch.def`

```

4301 \<*kernel>
4302 \let\bbl@onlyswitch\@empty
4303 \input babel.def
4304 \let\bbl@onlyswitch\@undefined
4305 \</kernel>
4306 %
4307 % \section{Error messages}
4308 %
4309 % They are loaded when |\bbl@error| is first called. To save space, the
4310 % main code just identifies them with a tag, and messages are stored in
4311 % a separate file. Since it can be loaded anywhere, you make sure some
4312 % catcodes have the right value, although those for |\, |`, |^M|,
4313 % |%| and |=| are reset before loading the file.

```

```

4314 %
4315 (*errors)
4316 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4317 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4318 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4319 \catcode`\@=11 \catcode`\^=7
4320 %
4321 \ifx\MessageBreak\undefined
4322 \gdef\bbl@error@i#1#2{%
4323 \begingroup
4324 \newlinechar=`^^J
4325 \def\{^^J(babel) }%
4326 \errhelp{#2}\errmessage{\{#1}%
4327 \endgroup}
4328 \else
4329 \gdef\bbl@error@i#1#2{%
4330 \begingroup
4331 \def\{MessageBreak}%
4332 \PackageError{babel}{#1}{#2}%
4333 \endgroup}
4334 \fi
4335 \def\bbl@errmessage#1#2#3{%
4336 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4337 \bbl@error@i{#2}{#3}}
4338 % Implicit #2#3#4:
4339 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4340 %
4341 \bbl@errmessage{not-yet-available}
4342 {Not yet available}%
4343 {Find an armchair, sit down and wait}
4344 \bbl@errmessage{bad-package-option}%
4345 {Bad option '#1=#2'. Either you have misspelled the\\%
4346 key or there is a previous setting of '#1'. Valid\\%
4347 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4348 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4349 {See the manual for further details.}
4350 \bbl@errmessage{base-on-the-fly}
4351 {For a language to be defined on the fly 'base'\\%
4352 is not enough, and the whole package must be\\%
4353 loaded. Either delete the 'base' option or\\%
4354 request the languages explicitly}%
4355 {See the manual for further details.}
4356 \bbl@errmessage{undefined-language}
4357 {You haven't defined the language '#1' yet.\\%
4358 Perhaps you misspelled it or your installation\\%
4359 is not complete}%
4360 {Your command will be ignored, type <return> to proceed}
4361 \bbl@errmessage{shorthand-is-off}
4362 {I can't declare a shorthand turned off (\string#2)}
4363 {Sorry, but you can't use shorthands which have been\\%
4364 turned off in the package options}
4365 \bbl@errmessage{not-a-shorthand}
4366 {The character '\string #1' should be made a shorthand character;\\%
4367 add the command \string\usesshorthands\string{#1\string} to
4368 the preamble.\\%
4369 I will ignore your instruction}%
4370 {You may proceed, but expect unexpected results}
4371 \bbl@errmessage{not-a-shorthand-b}
4372 {I can't switch '\string#2' on or off--not a shorthand}%
4373 {This character is not a shorthand. Maybe you made\\%
4374 a typing mistake? I will ignore your instruction.}
4375 \bbl@errmessage{unknown-attribute}
4376 {The attribute #2 is unknown for language #1.}%

```

```

4377 {Your command will be ignored, type <return> to proceed}
4378 \bbl@errmessage{missing-group}
4379 {Missing group for string \string#1}%
4380 {You must assign strings to some category, typically\\%
4381 captions or extras, but you set none}
4382 \bbl@errmessage{only-lua-xe}
4383 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4384 {Consider switching to these engines.}
4385 \bbl@errmessage{only-lua}
4386 {This macro is available only in LuaLaTeX.}%
4387 {Consider switching to that engine.}
4388 \bbl@errmessage{unknown-provide-key}
4389 {Unknown key '#1' in \string\babelprovide}%
4390 {See the manual for valid keys}%
4391 \bbl@errmessage{unknown-mapfont}
4392 {Option '\bbl@KVP@mapfont' unknown for\\%
4393 mapfont. Use 'direction'.}%
4394 {See the manual for details.}
4395 \bbl@errmessage{no-ini-file}
4396 {There is no ini file for the requested language\\%
4397 (#1: \language). Perhaps you misspelled it or your\\%
4398 installation is not complete.}%
4399 {Fix the name or reinstall babel.}
4400 \bbl@errmessage{digits-is-reserved}
4401 {The counter name 'digits' is reserved for mapping\\%
4402 decimal digits}%
4403 {Use another name.}
4404 \bbl@errmessage{limit-two-digits}
4405 {Currently two-digit years are restricted to the\\
4406 range 0-9999.}%
4407 {There is little you can do. Sorry.}
4408 \bbl@errmessage{alphabetic-too-large}
4409 {Alphabetic numeral too large (#1)}%
4410 {Currently this is the limit.}
4411 \bbl@errmessage{no-ini-info}
4412 {I've found no info for the current locale.\\%
4413 The corresponding ini file has not been loaded\\%
4414 Perhaps it doesn't exist}%
4415 {See the manual for details.}
4416 \bbl@errmessage{unknown-ini-field}
4417 {Unknown field '#1' in \string\BCPdata.\\%
4418 Perhaps you misspelled it.}%
4419 {See the manual for details.}
4420 \bbl@errmessage{unknown-locale-key}
4421 {Unknown key for locale '#2':\\%
4422 #3\\%
4423 \string#1 will be set to \relax}%
4424 {Perhaps you misspelled it.}%
4425 \bbl@errmessage{adjust-only-vertical}
4426 {Currently, #1 related features can be adjusted only\\%
4427 in the main vertical list.}%
4428 {Maybe things change in the future, but this is what it is.}
4429 \bbl@errmessage{layout-only-vertical}
4430 {Currently, layout related features can be adjusted only\\%
4431 in vertical mode.}%
4432 {Maybe things change in the future, but this is what it is.}
4433 \bbl@errmessage{bidi-only-lua}
4434 {The bidi method 'basic' is available only in\\%
4435 luatex. I'll continue with 'bidi=default', so\\%
4436 expect wrong results}%
4437 {See the manual for further details.}
4438 \bbl@errmessage{multiple-bidi}
4439 {Multiple bidi settings inside a group}%

```

```

4440 {I'll insert a new group, but expect wrong results.}
4441 \bbl@errmessage{unknown-package-option}
4442 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4443 or the language definition file \CurrentOption.ldf\\%
4444 was not found%
4445 \bbl@tempa}
4446 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4447 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4448 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4449 \bbl@errmessage{config-not-found}
4450 {Local config file '\bbl@opt@config.cfg' not found}%
4451 {Perhaps you misspelled it.}
4452 \bbl@errmessage{late-after-babel}
4453 {Too late for \string\AfterBabelLanguage}%
4454 {Languages have been loaded, so I can do nothing}
4455 \bbl@errmessage{double-hyphens-class}
4456 {Double hyphens aren't allowed in \string\babelcharclass\\%
4457 because it's potentially ambiguous}%
4458 {See the manual for further info}
4459 \bbl@errmessage{unknown-interchar}
4460 {'#1' for '\language' cannot be enabled.\\%
4461 Maybe there is a typo.}%
4462 {See the manual for further details.}
4463 \bbl@errmessage{unknown-interchar-b}
4464 {'#1' for '\language' cannot be disabled.\\%
4465 Maybe there is a typo.}%
4466 {See the manual for further details.}
4467 \bbl@errmessage{charproperty-only-vertical}
4468 {\string\babelcharproperty\space can be used only in\\%
4469 vertical mode (preamble or between paragraphs)}%
4470 {See the manual for further info}
4471 \bbl@errmessage{unknown-char-property}
4472 {No property named '#2'. Allowed values are\\%
4473 direction (bc), mirror (bmg), and linebreak (lb)}%
4474 {See the manual for further info}
4475 \bbl@errmessage{bad-transform-option}
4476 {Bad option '#1' in a transform.\\%
4477 I'll ignore it but expect more errors}%
4478 {See the manual for further info.}
4479 \bbl@errmessage{font-conflict-transforms}
4480 {Transforms cannot be re-assigned to different\\%
4481 fonts. The conflict is in '\bbl@kv@label'.\\%
4482 Apply the same fonts or use a different label}%
4483 {See the manual for further details.}
4484 \bbl@errmessage{transform-not-available}
4485 {'#1' for '\language' cannot be enabled.\\%
4486 Maybe there is a typo or it's a font-dependent transform}%
4487 {See the manual for further details.}
4488 \bbl@errmessage{transform-not-available-b}
4489 {'#1' for '\language' cannot be disabled.\\%
4490 Maybe there is a typo or it's a font-dependent transform}%
4491 {See the manual for further details.}
4492 \bbl@errmessage{year-out-range}
4493 {Year out of range.\\%
4494 The allowed range is #1}%
4495 {See the manual for further details.}
4496 \bbl@errmessage{only-pdftex-lang}
4497 {The '#1' ldf style doesn't work with #2,\\%
4498 but you can use the ini locale instead.\\%
4499 Try adding 'provide=*' to the option list. You may\\%
4500 also want to set 'bidi=' to some value.}%
4501 {See the manual for further details.}
4502 \</errors>

```



4503  $\langle$ \*patterns $\rangle$

## 7 Loading hyphenation patterns

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

```
4504  $\langle$ Make sure ProvidesFile is defined $\rangle$ 
4505 \ProvidesFile{hyphen.cfg}[ $\langle$ date $\rangle$ ] v $\langle$ version $\rangle$  Babel hyphens]
4506 \xdef\bbl@format{\jobname}
4507 \def\bbl@version{ $\langle$ version $\rangle$ }
4508 \def\bbl@date{ $\langle$ date $\rangle$ }
4509 \ifx\AtBeginDocument\undefined
4510   \def\@empty{}
4511 \fi
4512  $\langle$ Define core switching macros $\rangle$ 
```

$\backslash$ process@line Each line in the file `language.dat` is processed by  $\backslash$ 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  $\backslash$ process@synonym is called; otherwise the macro  $\backslash$ process@language will continue.

```
4513 \def\process@line#1#2 #3 #4 {%
4514   \ifx=#1%
4515     \process@synonym{#2}%
4516   \else
4517     \process@language{#1#2}{#3}{#4}%
4518   \fi
4519   \ignorespaces}
```

$\backslash$ 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.

```
4520 \toks@{}
4521 \def\bbl@languages{}
```

When no languages have been loaded yet, the name following the `=` will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The `\relax` just helps to the `\if` below catching synonyms without a language.) Otherwise the name will be a synonym for the language loaded last. We also need to copy the `hyphenmin` parameters for the synonym.

```
4522 \def\process@synonym#1{%
4523   \ifnum\last@language=\m@ne
4524     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4525   \else
4526     \expandafter\chardef\csname l@#1\endcsname\last@language
4527     \wlog{\string\l@#1=\string\language\the\last@language}%
4528     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4529       \csname\language\hyphenmins\endcsname
4530     \let\bbl@elt\relax
4531     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}%
4532   \fi}
```

$\backslash$ process@language The macro  $\backslash$ 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  $\backslash$ addlanguage to allocate a pattern register and to make that register ‘active’. Then the pattern file is read.

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

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

Pattern files may contain assignments to `\lefthyphenmin` and `\righthyphenmin`.  $\TeX$  does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\langle lang \rangle 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{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

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

```

4533 \def\process@language#1#2#3{%
4534   \expandafter\addlanguage\csname l@#1\endcsname
4535   \expandafter\language\csname l@#1\endcsname
4536   \edef\language#1{%
4537     \bbl@hook@everylanguage{#1}%
4538     % > luatex
4539     \bbl@get@enc#1:.\@@@
4540     \beginngroup
4541       \lefthyphenmin\m@ne
4542       \bbl@hook@loadpatterns{#2}%
4543       % > luatex
4544       \ifnum\lefthyphenmin=\m@ne
4545       \else
4546         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4547           \the\lefthyphenmin\the\righthyphenmin}%
4548       \fi
4549     \endgroup
4550   \def\bbl@tempa{#3}%
4551   \ifx\bbl@tempa\@empty\else
4552     \bbl@hook@loadexceptions{#3}%
4553     % > luatex
4554   \fi
4555   \let\bbl@elt\relax
4556   \edef\bbl@languages{%
4557     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4558   \ifnum\the\language=\z@
4559     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4560       \set@hyphenmins\tw@\thr@@\relax
4561     \else
4562       \expandafter\expandafter\expandafter\set@hyphenmins
4563       \csname #1hyphenmins\endcsname
4564     \fi
4565     \the\toks@
4566     \toks@{}%
4567   \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.

```

4568 \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.

```

4569 \def\bbl@hook@everylanguage#1{}
4570 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4571 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns

```

```

4572 \def\bbl@hook@loadkernel#1{%
4573   \def\addlanguage{\csname newlanguage\endcsname}%
4574   \def\adddialect##1##2{%
4575     \global\chardef##1##2\relax
4576     \wlog{\string##1 = a dialect from \string\language##2}}%
4577   \def\iflanguage##1{%
4578     \expandafter\ifx\csname l@##1\endcsname\relax
4579       \@nolanerr{##1}%
4580     \else
4581       \ifnum\csname l@##1\endcsname=\language
4582         \expandafter\expandafter\expandafter\@firstoftwo
4583       \else
4584         \expandafter\expandafter\expandafter\@secondoftwo
4585       \fi
4586     \fi}%
4587   \def\providehyphenmins##1##2{%
4588     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4589       \@namedef{##1hyphenmins}{##2}%
4590     \fi}%
4591   \def\set@hyphenmins##1##2{%
4592     \lefthyphenmin##1\relax
4593     \righthyphenmin##2\relax}%
4594   \def\selectlanguage{%
4595     \errhelp{Selecting a language requires a package supporting it}%
4596     \errmessage{Not loaded}}%
4597   \let\foreignlanguage\selectlanguage
4598   \let\otherlanguage\selectlanguage
4599   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4600   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4601     \def\setlocale{%
4602       \errhelp{Find an armchair, sit down and wait}%
4603       \errmessage{(babel) Not yet available}}%
4604     \let\uselocale\setlocale
4605     \let\locale\setlocale
4606     \let\selectlocale\setlocale
4607     \let\localename\setlocale
4608     \let\textlocale\setlocale
4609     \let\textlanguage\setlocale
4610     \let\languagetext\setlocale}
4611   \begingroup
4612   \def\AddBabelHook#1#2{%
4613     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4614       \def\next{\toks1}%
4615     \else
4616       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4617     \fi
4618     \next}
4619   \ifx\directlua\@undefined
4620     \ifx\XeTeXinputencoding\@undefined\else
4621       \input xebabel.def
4622     \fi
4623   \else
4624     \input luababel.def
4625   \fi
4626   \openin1 = babel-\bbl@format.cfg
4627   \ifeof1
4628   \else
4629     \input babel-\bbl@format.cfg\relax
4630   \fi
4631   \closein1
4632 \endgroup
4633 \bbl@hook@loadkernel{switch.def}

```

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

```
4634 \openin1 = language.dat
```

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

```
4635 \def\languagename{english}%
4636 \ifEOF1
4637   \message{I couldn't find the file language.dat,\space
4638             I will try the file hyphen.tex}
4639   \input hyphen.tex\relax
4640   \chardef\l@english\z@
4641 \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`.

```
4642   \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.

```
4643   \loop
4644     \endlinechar\m@ne
4645     \read1 to \bbl@line
4646     \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.

```
4647     \if T\ifEOF1F\fi T\relax
4648     \ifx\bbl@line@empty\else
4649       \edef\bbl@line{\bbl@line\space\space\space}%
4650       \expandafter\process@line\bbl@line\relax
4651     \fi
4652   \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.

```
4653   \begingroup
4654     \def\bbl@elt#1#2#3#4{%
4655       \global\language=#2\relax
4656       \gdef\languagename{#1}%
4657       \def\bbl@elt##1##2##3##4{}}%
4658   \bbl@languages
4659 \endgroup
4660 \fi
4661 \closein1
```

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

```
4662 \if/\the\toks@/\else
4663   \errhelp{language.dat loads no language, only synonyms}
4664   \errmessage{Orphan language synonym}
4665 \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.

```
4666 \let\bbl@line\undefined
4667 \let\process@line\undefined
4668 \let\process@synonym\undefined
4669 \let\process@language\undefined
4670 \let\bbl@get@enc\undefined
4671 \let\bbl@hyph@enc\undefined
```

```

4672 \let\bbl@tempa\@undefined
4673 \let\bbl@hook@loadkernel\@undefined
4674 \let\bbl@hook@everylanguage\@undefined
4675 \let\bbl@hook@loadpatterns\@undefined
4676 \let\bbl@hook@loadexceptions\@undefined
4677 \</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].

```

4678 <<*More package options>> ≡
4679 \chardef\bbl@bidimode\z@
4680 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4681 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4682 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4683 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4684 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4685 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4686 <</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.

```

4687 <<*Font selection>> ≡
4688 \bbl@trace{Font handling with fontspec}
4689 \ifx\ExplSyntaxOn\@undefined\else
4690   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4691     \in@{,#1,}{,no-script,language-not-exist,}%
4692     \ifin@ \else\bbl@tempfs@nx{#1}{#2}\fi}
4693   \def\bbl@fs@warn@nxx#1#2#3{%
4694     \in@{,#1,}{,no-script,language-not-exist,}%
4695     \ifin@ \else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4696   \def\bbl@loadfontspec{%
4697     \let\bbl@loadfontspec\relax
4698     \ifx\fontspec\@undefined
4699       \usepackage{fontspec}%
4700       \fi}%
4701 \fi
4702 \@onlypreamble\babelfont
4703 \newcommand\babelfont[2][{}% 1=langs/scripts 2=fam
4704   \bbl@foreach{#1}{%
4705     \expandafter\ifx\csname date##1\endcsname\relax
4706       \IfFileExists{babel-##1.tex}%
4707         {\babelprovide{##1}}%
4708         {}%
4709     \fi}%
4710   \edef\bbl@tempa{#1}%
4711   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4712   \bbl@loadfontspec
4713   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4714   \bbl@bblfont}
4715 \newcommand\babelfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
4716   \bbl@ifunset{\bbl@tempb family}%
4717   {\bbl@providfam{\bbl@tempb}}%
4718   {}%
4719   % For the default font, just in case:
4720   \bbl@ifunset{\bbl@lsys@\language name}{\bbl@provide@lsys{\language name}}{}%

```

```

4721 \expandafter\bb@ifblank\expandafter{\bb@tempa}%
4722 {\bb@csarg\edef{\bb@tempb dflt@{<{#1}{#2}}}% save \bb@rmdflt@
4723 \bb@exp{%
4724 \let<\bb@tempb dflt@\language name>\<\bb@tempb dflt@>%
4725 \\\bb@font@set<\bb@tempb dflt@\language name>%
4726 \<\bb@tempb default>\<\bb@tempb family>}}%
4727 {\bb@foreach\bb@tempa{% ie \bb@rmdflt@lang / *scrt
4728 \bb@csarg\def{\bb@tempb dflt@##1}{<{#1}{#2}}}}%

```

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

```

4729 \def\bb@providefam#1{%
4730 \bb@exp{%
4731 \\\newcommand<#1default>{% Just define it
4732 \\\bb@add@list\\bb@font@fams{#1}%
4733 \\\DeclareRobustCommand<#1family>{%
4734 \\\not@math@alphabet<#1family>\relax
4735 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4736 \\\fontfamily<#1default>%
4737 \<ifx>\\\UseHooks\\@undefined<else>\\\UseHook{#1family}\<fi>%
4738 \\\selectfont}%
4739 \\\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.

```

4740 \def\bb@nostdfont#1{%
4741 \bb@ifunset{\bb@WFF@f@family}%
4742 {\bb@csarg\gdef{WFF@f@family}}% Flag, to avoid dupl warns
4743 \bb@infowarn{The current font is not a babel standard family:\\%
4744 #1%
4745 \fontname\font\\%
4746 There is nothing intrinsically wrong with this warning, and\\%
4747 you can ignore it altogether if you do not need these\\%
4748 families. But if they are used in the document, you should be\\%
4749 aware 'babel' will not set Script and Language for them, so\\%
4750 you may consider defining a new family with \string\babelfont.\\%
4751 See the manual for further details about \string\babelfont.\\%
4752 Reported}}
4753 {}}%
4754 \gdef\bb@switchfont{%
4755 \bb@ifunset{\bb@lsys@\language name}{\bb@provide@lsys{\language name}}}%
4756 \bb@exp{% eg Arabic -> arabic
4757 \lowercase{\edef\\bb@tempa{\bb@cl{sname}}}}%
4758 \bb@foreach\bb@font@fams{%
4759 \bb@ifunset{\bb@##1dflt@\language name}% (1) language?
4760 {\bb@ifunset{\bb@##1dflt*\\bb@tempa}% (2) from script?
4761 {\bb@ifunset{\bb@##1dflt}% 2=F - (3) from generic?
4762 {}% 123=F - nothing!
4763 {\bb@exp{% 3=T - from generic
4764 \global\let<\bb@##1dflt@\language name>%
4765 \<\bb@##1dflt@>}}}%
4766 {\bb@exp{% 2=T - from script
4767 \global\let<\bb@##1dflt@\language name>%
4768 \<\bb@##1dflt*\\bb@tempa>}}}%
4769 {}% 1=T - language, already defined
4770 \def\bb@tempa{\bb@nostdfont}}% TODO. Don't use \bb@tempa
4771 \bb@foreach\bb@font@fams{% don't gather with prev for
4772 \bb@ifunset{\bb@##1dflt@\language name}%
4773 {\bb@cs{famrst@##1}%
4774 \global\bb@csarg\let{famrst@##1}\relax}%
4775 {\bb@exp{% order is relevant. TODO: but sometimes wrong!
4776 \\\bb@add\\originalTeX{%
4777 \\\bb@font@rst{\bb@cl{##1dflt}}%
4778 \<##1default>\<##1family>{##1}}%
4779 \\\bb@font@set<\bb@##1dflt@\language name>% the main part!

```

```

4780 \<##ldefault>\<##lfamily>}}}%
4781 \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`.

```

4782 \ifx\fb@family\@undefined\else % if latex
4783 \ifcase\bbl@engine % if pdftex
4784 \let\bbl@cckstdfont\relax
4785 \else
4786 \def\bbl@cckstdfont{%
4787 \begingroup
4788 \global\let\bbl@cckstdfont\relax
4789 \let\bbl@tempa\@empty
4790 \bbl@foreach\bbl@font@fams{%
4791 \bbl@ifunset\bbl@##ldflt@}%
4792 {\@nameuse{##lfamily}}%
4793 \bbl@csarg\gdef{WFF@fb@family}}}% Flag
4794 \bbl@exp{\bbl@add\bbl@tempa{* \<##lfamily>= fb@family\\}%
4795 \space\space\fontname\font\\}}}%
4796 \bbl@csarg\xdef{##ldflt@}{fb@family}}%
4797 \expandafter\xdef\csname ##ldefault\endcsname{fb@family}}}%
4798 {}}%
4799 \ifx\bbl@tempa\@empty\else
4800 \bbl@infowarn{The following font families will use the default\\%
4801 settings for all or some languages:\\%
4802 \bbl@tempa
4803 There is nothing intrinsically wrong with it, but\\%
4804 'babel' will no set Script and Language, which could\\%
4805 be relevant in some languages. If your document uses\\%
4806 these families, consider redefining them with \string\babelfont.\\%
4807 Reported}%
4808 \fi
4809 \endgroup}
4810 \fi
4811 \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,  $\text{\LaTeX}$  can select two different series (bx and b), for what is conceptually a single one. This can lead to problems when a single family requires several fonts, depending on the language, mainly because ‘substitutions’ with some combinations are not done consistently – sometimes `bx/sc` is the correct font, but sometimes points to `b/n`, even if `b/sc` exists. So, some substitutions are redefined (in a somewhat hackish way, by inspecting if the variant declaration contains `>ssub*`).

```

4812 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4813 \bbl@xin@{<>}{#1}%
4814 \ifin@
4815 \bbl@exp{\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4816 \fi
4817 \bbl@exp{%
4818 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4819 \\bbl@ifsamestring{#2}{fb@family}}%
4820 {\\#3%
4821 \\bbl@ifsamestring{fb@series}{bfdefault}{\\bfseries}}}%
4822 \let\\bbl@tempa\relax}%
4823 {}}}}
4824 % TODO - next should be global?, but even local does its job. I'm
4825 % still not sure -- must investigate:
4826 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4827 \let\bbl@tempe\bbl@mapselect
4828 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4829 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslashfamily/}}}%

```

```

4830 \let\bbl@mapselect\relax
4831 \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4832 \let#4\@empty           %      Make sure \renewfontfamily is valid
4833 \bbl@exp{%
4834   \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4835   \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbl@cl{sname}}}%
4836   {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4837   \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbl@cl{lname}}}%
4838   {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4839   \let\\bbl@temp@fs@nx<__fontspec_warning:nx>%
4840   \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4841   \let\\bbl@temp@fs@nxx<__fontspec_warning:nxx>%
4842   \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4843   \\renewfontfamily\\#4%
4844   [\bbl@cl{lsys},%
4845     \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4846     #2]}{#3}% ie \bbl@exp{..}{#3}
4847 \bbl@exp{%
4848   \let<__fontspec_warning:nx>\\bbl@temp@fs@nx
4849   \let<__fontspec_warning:nxx>\\bbl@temp@fs@nxx}%
4850 \begingroup
4851   #4%
4852   \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4853 \endgroup % TODO. Find better tests:
4854 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4855   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4856 \ifin@
4857   \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4858 \fi
4859 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4860   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4861 \ifin@
4862   \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4863 \fi
4864 \let#4\bbl@temp@fam
4865 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4866 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4867 \def\bbl@font@rst#1#2#3#4{%
4868   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4869 \def\bbl@font@fams{rm,sf,tt}
4870 <</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.

```

4871 <<{*Footnote changes}>> ≡
4872 \bbl@trace{Bidi footnotes}
4873 \ifnum\bbl@bidimode>\z@ % Any bidi=
4874   \def\bbl@footnote#1#2#3{%
4875     \ifnextchar[%
4876       {\bbl@footnote@o{#1}{#2}{#3}}%
4877       {\bbl@footnote@x{#1}{#2}{#3}}%
4878   \long\def\bbl@footnote@x#1#2#3#4{%
4879     \bgroup

```



```

4880 \select@language@x{\bbl@main@language}%
4881 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4882 \egroup}
4883 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4884 \bgroup
4885 \select@language@x{\bbl@main@language}%
4886 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4887 \egroup}
4888 \def\bbl@footnotetext#1#2#3{%
4889 \@ifnextchar[%
4890 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4891 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4892 \long\def\bbl@footnotetext@x#1#2#3#4{%
4893 \bgroup
4894 \select@language@x{\bbl@main@language}%
4895 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4896 \egroup}
4897 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4898 \bgroup
4899 \select@language@x{\bbl@main@language}%
4900 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4901 \egroup}
4902 \def\BabelFootnote#1#2#3#4{%
4903 \ifx\bbl@fn@footnote\undefined
4904 \let\bbl@fn@footnote\footnote
4905 \fi
4906 \ifx\bbl@fn@footnotetext\undefined
4907 \let\bbl@fn@footnotetext\footnotetext
4908 \fi
4909 \bbl@ifblank{#2}%
4910 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4911 \@namedef{\bbl@stripslash#1text}%
4912 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4913 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4914 \@namedef{\bbl@stripslash#1text}%
4915 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4916 \fi
4917 <</Footnote changes>>

```

Now, the code.

```

4918 (*xetex)
4919 \def\BabelStringsDefault{unicode}
4920 \let\xebbl@stop\relax
4921 \AddBabelHook{xetex}{encodedcommands}{%
4922 \def\bbl@tempa{#1}%
4923 \ifx\bbl@tempa@empty
4924 \XeTeXinputencoding"bytes"%
4925 \else
4926 \XeTeXinputencoding"#1"%
4927 \fi
4928 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4929 \AddBabelHook{xetex}{stopcommands}{%
4930 \xebbl@stop
4931 \let\xebbl@stop\relax}
4932 \def\bbl@intraspace#1 #2 #3@@{%
4933 \bbl@csarg\gdef{\xeisp@{language}}%
4934 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4935 \def\bbl@intrapenalty#1@@{%
4936 \bbl@csarg\gdef{\xeipn@{language}}%
4937 {\XeTeXlinebreakpenalty #1\relax}}
4938 \def\bbl@provide@intraspace{%
4939 \bbl@xin@{/s}{/bbl@c{l}{lnbrk}}%
4940 \ifin@else\bbl@xin@{/c}{/bbl@c{l}{lnbrk}}\fi

```

```

4941 \ifin@
4942   \bbl@ifunset{bbl@intsp@\language\name}{}%
4943   {\expandafter\ifx\csname bbl@intsp@\language\name\endcsname\@empty\else
4944     \ifx\bbl@KVP@intraspace\@nnil
4945       \bbl@exp{%
4946         \\bbl@intraspace\bbl@cl{intsp}\\@}%
4947       \fi
4948       \ifx\bbl@KVP@intrapenalty\@nnil
4949         \bbl@intrapenalty0\@@
4950       \fi
4951     \fi
4952     \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4953       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4954     \fi
4955     \ifx\bbl@KVP@intrapenalty\@nnil\else
4956       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4957     \fi
4958     \bbl@exp{%
4959       % TODO. Execute only once (but redundant):
4960       \\bbl@add\<extras\language\name>{%
4961         \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4962         \<bbl@xeisp@\language\name>%
4963         \<bbl@xeipn@\language\name>}%
4964       \\bbl@tglobal\<extras\language\name>%
4965       \\bbl@add\<noextras\language\name>{%
4966         \XeTeXlinebreaklocale ""}%
4967       \\bbl@tglobal\<noextras\language\name>}%
4968     \ifx\bbl@ispacesize\@undefined
4969       \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4970     \ifx\AtBeginDocument\@notprerr
4971       \expandafter\@secondoftwo % to execute right now
4972     \fi
4973     \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4974   \fi}%
4975 \fi}
4976 \ifx\DisableBabelHook\@undefined\endinput\fi
4977 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4978 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4979 \DisableBabelHook{babel-fontspec}
4980 <<Font selection>>
4981 \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.

```

4982 \ifnum\xe@alloc@intercharclass<\thr@@
4983   \xe@alloc@intercharclass\thr@@
4984 \fi
4985 \chardef\bbl@xe@class@default@=\z@
4986 \chardef\bbl@xe@class@cjkideogram@=\@ne
4987 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
4988 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
4989 \chardef\bbl@xe@class@boundary@=4095
4990 \chardef\bbl@xe@class@ignore@=4096

```

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

```

4991 \AddBabelHook{babel-interchar}{beforeextras}{%
4992   \@nameuse{\bbl@xechars@\language\name}}

```

```

4993 \DisableBabelHook{babel-interchar}
4994 \protected\def\bbl@charclass#1{%
4995   \ifnum\count@<\z@
4996     \count@-\count@
4997     \loop
4998       \bbl@exp{%
4999         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5000         \XeTeXcharclass\count@ \bbl@tempc
5001         \ifnum\count@<`#1\relax
5002         \advance\count@\@ne
5003       \repeat
5004   \else
5005     \babel@savevariable{\XeTeXcharclass`#1}%
5006     \XeTeXcharclass`#1 \bbl@tempc
5007   \fi
5008   \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, \}). As a special case, hyphens are stored as \bbl@upto, to deal with ranges.

```

5009 \newcommand\IfBabelIntercharT[1]{%
5010   \let\bbl@tempa\@gobble           % Assume to ignore
5011   \edef\bbl@tempb{\zap@space#1 \@empty}%
5012   \ifx\bbl@KVP@interchar\@nnil\else
5013     \bbl@replace\bbl@KVP@interchar{ }{,}%
5014     \bbl@foreach\bbl@tempb{%
5015       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5016     \ifin@
5017       \let\bbl@tempa\@firstofone
5018     \fi}%
5019   \fi
5020   \bbl@tempa}
5021 \newcommand\babelcharclass[3]{%
5022   \EnableBabelHook{babel-interchar}%
5023   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5024   \def\bbl@tempb##1{%
5025     \ifx##1\@empty\else
5026       \ifx##1-
5027         \bbl@upto
5028       \else
5029         \bbl@charclass{%
5030           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5031       \fi
5032       \expandafter\bbl@tempb
5033     \fi}%
5034   \bbl@ifunset{\bbl@xechars@#1}%
5035   {\toks@{%
5036     \babel@savevariable\XeTeXinterchartokenstate
5037     \XeTeXinterchartokenstate\@ne
5038   }}%
5039   {\toks@\expandafter\expandafter\expandafter{%
5040     \csname bbl@xechars@#1\endcsname}}%
5041   \bbl@csarg\edef{xechars@#1}{%
5042     \the\toks@
5043     \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5044     \bbl@tempb#3\@empty}}
5045 \protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
5046 \protected\def\bbl@upto{%
5047   \ifnum\count@>\z@
5048     \advance\count@\@ne

```

```

5049 \count@-\count@
5050 \else\ifnum\count@=\z@
5051 \bbl@charclass{-}%
5052 \else
5053 \bbl@error{double-hyphens-class}{\}\}\}%
5054 \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>`.

```

5055 \newcommand\babelinterchar[5][\]{%
5056 \let\bbl@kv@label\@empty
5057 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}}%
5058 \@namedef{\zap@space\bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5059 {\ifnum\language=\l@nohyphenation
5060 \expandafter\@gobble
5061 \else
5062 \expandafter\@firstofone
5063 \fi
5064 {#5}}}%
5065 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5066 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}}%
5067 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}}%
5068 \XeTeXinterchartoks
5069 \@nameuse{bbl@xeclass@\bbl@tempa @#%
5070 \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{\}\}\} %
5071 \@nameuse{bbl@xeclass@\bbl@tempb @#%
5072 \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{\}\}\} %
5073 = \expandafter{%
5074 \csname\bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5075 \csname\zap@space\bbl@xeinter@\bbl@kv@label
5076 @#3@#4@#2 \@empty\endcsname}}}%
5077 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5078 \bbl@ifunset{bbl@ic@#1@\language}%
5079 {\bbl@error{unknown-interchar}{#1}{\}\}\}%
5080 {\bbl@csarg\let{ic@#1@\language}\@firstofone}}
5081 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5082 \bbl@ifunset{bbl@ic@#1@\language}%
5083 {\bbl@error{unknown-interchar-b}{#1}{\}\}\}%
5084 {\bbl@csarg\let{ic@#1@\language}\@gobble}}
5085 \xetex)

```

## 10.1 Layout

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

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

`\advance\bbl@startskip\adim`, `\bbl@startskip\adim`.

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

```

5086 (*xetex | texxet)
5087 \providecommand\bbl@provide@intraspace{}
5088 \bbl@trace{Redefinitions for bidi layout}
5089 \def\bbl@sspre@caption{%
5090 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
5091 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5092 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5093 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5094 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
5095 \def\@hangfrom#1{%
5096 \setbox\@tempboxa\hbox{#1}}%
5097 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5098 \noindent\box\@tempboxa}

```

```

5099 \def\raggedright{%
5100   \let\\\@centercr
5101   \bbl@startskip\z@skip
5102   \@rightskip\@flushglue
5103   \bbl@endskip\@rightskip
5104   \parindent\z@
5105   \parfillskip\bbl@startskip}
5106 \def\raggedleft{%
5107   \let\\\@centercr
5108   \bbl@startskip\@flushglue
5109   \bbl@endskip\z@skip
5110   \parindent\z@
5111   \parfillskip\bbl@endskip}
5112 \fi
5113 \IfBabelLayout{lists}
5114   {\bbl@sreplace\list
5115     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5116     \def\bbl@listleftmargin{%
5117       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5118     \ifcase\bbl@engine
5119       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5120       \def\p@enumiii{\p@enumii}\theenumii{}\fi
5121     \fi
5122     \bbl@sreplace\@verbatim
5123       {\leftskip\@totalleftmargin}%
5124       {\bbl@startskip\textwidth
5125         \advance\bbl@startskip-\linewidth}%
5126     \bbl@sreplace\@verbatim
5127       {\rightskip\z@skip}%
5128       {\bbl@endskip\z@skip}}%
5129   {}
5130 \IfBabelLayout{contents}
5131   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5132     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5133   {}
5134 \IfBabelLayout{columns}
5135   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5136     \def\bbl@outputbox#1{%
5137       \hb@xt@\textwidth{%
5138         \hskip\columnwidth
5139         \hfil
5140         {\normalcolor\vrule \@width\columnseprule}%
5141         \hfil
5142         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5143         \hskip-\textwidth
5144         \hb@xt@\columnwidth{\box\@outputbox \hss}%
5145         \hskip\columnsep
5146         \hskip\columnwidth}}}%
5147   {}
5148 \langle\langle Footnote changes\rangle\rangle
5149 \IfBabelLayout{footnotes}%
5150   {\BabelFootnote\footnote\languagename{}\{}}%
5151   \BabelFootnote\localfootnote\languagename{}\{}}%
5152   \BabelFootnote\mainfootnote{}\{}}%
5153   {}
5154 \IfBabelLayout{counters*}%
5155   {\bbl@add\bbl@opt@layout{.counters.}%
5156     \AddToHook{shipout/before}{%
5157       \let\bbl@tempa\babelsublr
5158       \let\babelsublr\@firstofone

```

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.

```

5159 \let\bbl@save@thepage\thepage
5160 \protected@edef\thepage{\thepage}%
5161 \let\babelsublr\bbl@tempa}%
5162 \AddToHook{shipout/after}{%
5163 \let\thepage\bbl@save@thepage}}{}
5164 \IfBabelLayout{counters}%
5165 {\let\bbl@latinarabic=\@arabic
5166 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5167 \let\bbl@asciroman=\@roman
5168 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5169 \let\bbl@asciiRoman=\@Roman
5170 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5171 \fi % end if layout
5172 \</xetex>|texet)

```

## 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).

```

5173 \<texet>
5174 \def\bbl@provide@extra#1{%
5175 % == auto-select encoding ==
5176 \ifx\bbl@encoding@select@off\@empty\else
5177 \bbl@ifunset{\bbl@encoding@#1}%
5178 {\def\@elt##1{,##1,}%
5179 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5180 \count@\z@
5181 \bbl@foreach\bbl@tempe{%
5182 \def\bbl@tempd{##1}% Save last declared
5183 \advance\count@\@ne}%
5184 \ifnum\count@>\@ne % (1)
5185 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5186 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5187 \bbl@replace\bbl@tempa{ },}%
5188 \global\bbl@csarg\let{encoding@#1}\@empty
5189 \bbl@xin@{\bbl@tempd,}{,\bbl@tempa,}%
5190 \ifin\@else % if main encoding included in ini, do nothing
5191 \let\bbl@tempb\relax
5192 \bbl@foreach\bbl@tempa{%
5193 \ifx\bbl@tempb\relax
5194 \bbl@xin@{,##1,}{,\bbl@tempe,}%
5195 \ifin\def\bbl@tempb{##1}\fi
5196 \fi}%
5197 \ifx\bbl@tempb\relax\else
5198 \bbl@exp{%
5199 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>}%
5200 \gdef\<\bbl@encoding@#1>{%
5201 \\\babel@save\\f@encoding
5202 \\\bbl@add\\originalTeX{\\selectfont}%
5203 \\\fontencoding{\bbl@tempb}%
5204 \\\selectfont}}%
5205 \fi
5206 \fi
5207 \fi}%
5208 {}%
5209 \fi}
5210 \</texet>

```

## 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`).

```

5211 (*luatex)
5212 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5213 \bbl@trace{Read language.dat}
5214 \ifx\bbl@readstream\undefined
5215   \csname newread\endcsname\bbl@readstream
5216 \fi
5217 \begingroup
5218   \toks@{}
5219   \count@z@ % 0=start, 1=0th, 2=normal
5220   \def\bbl@process@line#1#2 #3 #4 {%
5221     \ifx=#1%
5222       \bbl@process@synonym{#2}%
5223     \else
5224       \bbl@process@language{#1#2}{#3}{#4}%
5225     \fi
5226     \ignorespaces}
5227 \def\bbl@manylang{%
5228   \ifnum\bbl@last>\@ne
5229     \bbl@info{Non-standard hyphenation setup}%
5230   \fi
5231   \let\bbl@manylang\relax}
5232 \def\bbl@process@language#1#2#3{%
5233   \ifcase\count@
5234     \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
5235   \or
5236     \count@\tw@
5237   \fi
5238   \ifnum\count@=\tw@
5239     \expandafter\addlanguage\csname l@#1\endcsname
5240     \language\allocationnumber
5241     \chardef\bbl@last\allocationnumber
5242     \bbl@manylang
5243     \let\bbl@elt\relax

```

```

5244 \xdef\bbbl@languages{%
5245 \bbbl@languages\bbbl@elt{#1}{\the\language}{#2}{#3}}%
5246 \fi
5247 \the\toks@
5248 \toks@{}}
5249 \def\bbbl@process@synonym@aux#1#2{%
5250 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5251 \let\bbbl@elt\relax
5252 \xdef\bbbl@languages{%
5253 \bbbl@languages\bbbl@elt{#1}{#2}{}}}%
5254 \def\bbbl@process@synonym#1{%
5255 \ifcase\count@
5256 \toks@\expandafter{\the\toks@\relax\bbbl@process@synonym{#1}}%
5257 \or
5258 \ifundefined{zth@#1}{\bbbl@process@synonym@aux{#1}{0}}}%
5259 \else
5260 \bbbl@process@synonym@aux{#1}{\the\bbbl@last}%
5261 \fi}
5262 \ifx\bbbl@languages\undefined % Just a (sensible?) guess
5263 \chardef\l@english\z@
5264 \chardef\l@USenglish\z@
5265 \chardef\bbbl@last\z@
5266 \global\@namedef{bbbl@hyphendata@0}{{hyphen.tex}}
5267 \gdef\bbbl@languages{%
5268 \bbbl@elt{english}{0}{hyphen.tex}}%
5269 \bbbl@elt{USenglish}{0}{}}
5270 \else
5271 \global\let\bbbl@languages@format\bbbl@languages
5272 \def\bbbl@elt#1#2#3#4{% Remove all except language 0
5273 \ifnum#2>\z@\else
5274 \noexpand\bbbl@elt{#1}{#2}{#3}{#4}%
5275 \fi}%
5276 \xdef\bbbl@languages{\bbbl@languages}%
5277 \fi
5278 \def\bbbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5279 \bbbl@languages
5280 \openin\bbbl@readstream=language.dat
5281 \ifeof\bbbl@readstream
5282 \bbbl@warning{I couldn't find language.dat. No additional\\%
5283 patterns loaded. Reported}%
5284 \else
5285 \loop
5286 \endlinechar\m@ne
5287 \read\bbbl@readstream to \bbbl@line
5288 \endlinechar``^^M
5289 \if T\ifeof\bbbl@readstream F\fi T\relax
5290 \ifx\bbbl@line\empty\else
5291 \edef\bbbl@line{\bbbl@line\space\space\space}%
5292 \expandafter\bbbl@process@line\bbbl@line\relax
5293 \fi
5294 \repeat
5295 \fi
5296 \closein\bbbl@readstream
5297 \endgroup
5298 \bbbl@trace{Macros for reading patterns files}
5299 \def\bbbl@get@enc#1:#2:#3@@{\def\bbbl@hyph@enc{#2}}
5300 \ifx\babelcatcodetablenum\undefined
5301 \ifx\newcatcodetable\undefined
5302 \def\babelcatcodetablenum{5211}
5303 \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5304 \else
5305 \newcatcodetable\babelcatcodetablenum
5306 \newcatcodetable\bbbl@pattcodes

```



```

5307 \fi
5308 \else
5309 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5310 \fi
5311 \def\bbl@luapatterns#1#2{%
5312 \bbl@get@enc#1:.\@@@
5313 \setbox\z@\hbox\bgroup
5314 \begingroup
5315 \savecatcodetable\babelcatcodetablenum\relax
5316 \initcatcodetable\bbl@pattcodes\relax
5317 \catcodetable\bbl@pattcodes\relax
5318 \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5319 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5320 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5321 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\. =12
5322 \catcode\- =12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5323 \catcode\`=12 \catcode\'=12 \catcode\"=12
5324 \input #1\relax
5325 \catcodetable\babelcatcodetablenum\relax
5326 \endgroup
5327 \def\bbl@tempa{#2}%
5328 \ifx\bbl@tempa\@empty\else
5329 \input #2\relax
5330 \fi
5331 \egroup}%
5332 \def\bbl@patterns@lua#1{%
5333 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5334 \csname l@#1\endcsname
5335 \edef\bbl@tempa{#1}%
5336 \else
5337 \csname l@#1:f@encoding\endcsname
5338 \edef\bbl@tempa{#1:f@encoding}%
5339 \fi\relax
5340 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5341 \@ifundefined{bbl@hyphendata@the\language}%
5342 {\def\bbl@elt##1##2##3##4{%
5343 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5344 \def\bbl@tempb{##3}%
5345 \ifx\bbl@tempb\@empty\else % if not a synonymous
5346 \def\bbl@tempc{{##3}{##4}}}%
5347 \fi
5348 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5349 \fi}%
5350 \bbl@languages
5351 \@ifundefined{bbl@hyphendata@the\language}%
5352 {\bbl@info{No hyphenation patterns were set for\\%
5353 language '\bbl@tempa'. Reported}}%
5354 {\expandafter\expandafter\expandafter\bbl@luapatterns
5355 \csname bbl@hyphendata@the\language\endcsname}}}}
5356 \endinput\fi
5357 % Here ends \ifx\AddBabelHook\@undefined
5358 % A few lines are only read by hyphen.cfg
5359 \ifx\DisableBabelHook\@undefined
5360 \AddBabelHook{luatex}{everylanguage}{%
5361 \def\process@language##1##2##3{%
5362 \def\process@line####1####2 ####3 ####4 {}}}
5363 \AddBabelHook{luatex}{loadpatterns}{%
5364 \input #1\relax
5365 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5366 {{#1}}}}
5367 \AddBabelHook{luatex}{loadexceptions}{%
5368 \input #1\relax
5369 \def\bbl@tempb##1##2{{##1}{#1}}}%

```

```

5370 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5371 {\expandafter\expandafter\expandafter\bbl@tempb
5372 \csname bbl@hyphendata@the\language\endcsname}}
5373 \endinput\fi
5374 % Here stops reading code for hyphen.cfg
5375 % The following is read the 2nd time it's loaded
5376 \beginingroup % TODO - to a lua file
5377 \catcode`\%=12
5378 \catcode`\'=12
5379 \catcode`\ "=12
5380 \catcode`\:=12
5381 \directlua{
5382   Babel = Babel or {}
5383   function Babel.bytes(line)
5384     return line:gsub("(.)",
5385       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5386   end
5387   function Babel.begin_process_input()
5388     if luatexbase and luatexbase.add_to_callback then
5389       luatexbase.add_to_callback('process_input_buffer',
5390         Babel.bytes, 'Babel.bytes')
5391     else
5392       Babel.callback = callback.find('process_input_buffer')
5393       callback.register('process_input_buffer', Babel.bytes)
5394     end
5395   end
5396   function Babel.end_process_input ()
5397     if luatexbase and luatexbase.remove_from_callback then
5398       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5399     else
5400       callback.register('process_input_buffer', Babel.callback)
5401     end
5402   end
5403   function Babel.addpatterns(pp, lg)
5404     local lg = lang.new(lg)
5405     local pats = lang.patterns(lg) or ''
5406     lang.clear_patterns(lg)
5407     for p in pp:gmatch('[^s]+') do
5408       ss = ''
5409       for i in string.utfcharacters(p:gsub('%d', '')) do
5410         ss = ss .. '%d?' .. i
5411       end
5412       ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5413       ss = ss:gsub('%.%d%?$', '%%.')
5414       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5415       if n == 0 then
5416         tex.sprint(
5417           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5418           .. p .. [[{ }]])
5419         pats = pats .. ' ' .. p
5420       else
5421         tex.sprint(
5422           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5423           .. p .. [[{ }]])
5424       end
5425     end
5426     lang.patterns(lg, pats)
5427   end
5428   Babel.characters = Babel.characters or {}
5429   Babel.ranges = Babel.ranges or {}
5430   function Babel.hlist_has_bidi(head)
5431     local has_bidi = false
5432     local ranges = Babel.ranges

```

```

5433     for item in node.traverse(head) do
5434         if item.id == node.id'glyph' then
5435             local itemchar = item.char
5436             local chardata = Babel.characters[itemchar]
5437             local dir = chardata and chardata.d or nil
5438             if not dir then
5439                 for nn, et in ipairs(ranges) do
5440                     if itemchar < et[1] then
5441                         break
5442                     elseif itemchar <= et[2] then
5443                         dir = et[3]
5444                         break
5445                     end
5446                 end
5447             end
5448             if dir and (dir == 'al' or dir == 'r') then
5449                 has_bidi = true
5450             end
5451         end
5452     end
5453     return has_bidi
5454 end
5455 function Babel.set_chranges_b (script, chrng)
5456     if chrng == '' then return end
5457     texio.write('Replacing ' .. script .. ' script ranges')
5458     Babel.script_blocks[script] = {}
5459     for s, e in string.gmatch(chrng..' ', '(.-%.-%.-%s)') do
5460         table.insert(
5461             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5462     end
5463 end
5464 function Babel.discard_sublr(str)
5465     if str:find( [[\string\indexentry]] ) and
5466        str:find( [[\string\babelsublr]] ) then
5467         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5468                        function(m) return m:sub(2,-2) end )
5469     end
5470     return str
5471 end
5472 }
5473 \endgroup
5474 \ifx\newattribute\undefined\else % Test for plain
5475     \newattribute\bbl@attr@locale
5476     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5477     \AddBabelHook{luatex}{beforeextras}{%
5478         \setattribute\bbl@attr@locale\localeid}
5479 \fi
5480 \def\BabelStringsDefault{unicode}
5481 \let\luabbl@stop\relax
5482 \AddBabelHook{luatex}{encodedcommands}{%
5483     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5484     \ifx\bbl@tempa\bbl@tempb\else
5485         \directlua{Babel.begin_process_input()}%
5486         \def\luabbl@stop{%
5487             \directlua{Babel.end_process_input()}}%
5488     \fi}%
5489 \AddBabelHook{luatex}{stopcommands}{%
5490     \luabbl@stop
5491     \let\luabbl@stop\relax}
5492 \AddBabelHook{luatex}{patterns}{%
5493     \ifundefined{bbl@hyphendata@the\language}%
5494     { \def\bbl@elt##1##2##3##4{%
5495         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...

```

```

5496 \def\bbl@tempb{##3}%
5497 \ifx\bbl@tempb\@empty\else % if not a synonymous
5498 \def\bbl@tempc{{##3}{##4}}}%
5499 \fi
5500 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5501 \fi}%
5502 \bbl@languages
5503 \@ifundefined{bbl@hyphendata@the\language}%
5504 {\bbl@info{No hyphenation patterns were set for\\%
5505 language '#2'. Reported}}%
5506 {\expandafter\expandafter\expandafter\bbl@luapatterns
5507 \csname bbl@hyphendata@the\language\endcsname}}}%
5508 \@ifundefined{bbl@patterns@}{}%
5509 \begingroup
5510 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5511 \ifin@else
5512 \ifx\bbl@patterns@\@empty\else
5513 \directlua{ Babel.addpatterns(
5514 [[\bbl@patterns@]], \number\language) }%
5515 \fi
5516 \@ifundefined{bbl@patterns@#1}%
5517 \@empty
5518 {\directlua{ Babel.addpatterns(
5519 [[\space\csname bbl@patterns@#1\endcsname]],
5520 \number\language) }}%
5521 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5522 \fi
5523 \endgroup}%
5524 \bbl@exp{%
5525 \bbl@ifunset{bbl@prehc@\languagename}{}%
5526 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5527 {\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.

```

5528 \@onlypreamble\babelpatterns
5529 \AtEndOfPackage{%
5530 \newcommand\babelpatterns[2][\@empty]{%
5531 \ifx\bbl@patterns@relax
5532 \let\bbl@patterns@\@empty
5533 \fi
5534 \ifx\bbl@pttnlist@\@empty\else
5535 \bbl@warning{%
5536 You must not intermingle \string\selectlanguage\space and\\%
5537 \string\babelpatterns\space or some patterns will not\\%
5538 be taken into account. Reported}%
5539 \fi
5540 \ifx\@empty#1%
5541 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5542 \else
5543 \edef\bbl@tempb{\zap@space#1 \@empty}%
5544 \bbl@for\bbl@tempa\bbl@tempb{%
5545 \bbl@fixname\bbl@tempa
5546 \bbl@iflanguage\bbl@tempa{%
5547 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5548 \@ifundefined{bbl@patterns@\bbl@tempa}%
5549 \@empty
5550 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5551 #2}}}%
5552 \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.

```
5553% TODO - to a lua file
5554\directlua{
5555  Babel = Babel or {}
5556  Babel.linebreaking = Babel.linebreaking or {}
5557  Babel.linebreaking.before = {}
5558  Babel.linebreaking.after = {}
5559  Babel.locale = {} % Free to use, indexed by \localeid
5560  function Babel.linebreaking.add_before(func, pos)
5561    tex.print([[noexpand\csname bbl@lua-hyphenate\endcsname]])
5562    if pos == nil then
5563      table.insert(Babel.linebreaking.before, func)
5564    else
5565      table.insert(Babel.linebreaking.before, pos, func)
5566    end
5567  end
5568  function Babel.linebreaking.add_after(func)
5569    tex.print([[noexpand\csname bbl@lua-hyphenate\endcsname]])
5570    table.insert(Babel.linebreaking.after, func)
5571  end
5572}
5573\def\bbl@intraspace#1 #2 #3\@@{%
5574  \directlua{
5575    Babel = Babel or {}
5576    Babel.intraspaces = Babel.intraspaces or {}
5577    Babel.intraspaces['\csname bbl@sbc-p@\language-name\endcsname'] = %
5578      {b = #1, p = #2, m = #3}
5579    Babel.locale_props[\the\localeid].intraspace = %
5580      {b = #1, p = #2, m = #3}
5581  }}
5582\def\bbl@intrapenalty#1\@@{%
5583  \directlua{
5584    Babel = Babel or {}
5585    Babel.intrapenalties = Babel.intrapenalties or {}
5586    Babel.intrapenalties['\csname bbl@sbc-p@\language-name\endcsname'] = #1
5587    Babel.locale_props[\the\localeid].intrapenalty = #1
5588  }}
5589\begingroup
5590\catcode`\%=12
5591\catcode`\^=14
5592\catcode`\'=12
5593\catcode`\~=12
5594\gdef\bbl@seaintraspace{^
5595  \let\bbl@seaintraspace\relax
5596  \directlua{
5597    Babel = Babel or {}
5598    Babel.sea_enabled = true
5599    Babel.sea_ranges = Babel.sea_ranges or {}
5600    function Babel.set_chranges (script, chrng)
5601      local c = 0
5602      for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5603        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5604        c = c + 1
5605      end
5606    end
5607    function Babel.sea_disc_to_space (head)
5608      local sea_ranges = Babel.sea_ranges
5609      local last_char = nil
```

```

5610     local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5611     for item in node.traverse(head) do
5612         local i = item.id
5613         if i == node.id'glyph' then
5614             last_char = item
5615         elseif i == 7 and item.subtype == 3 and last_char
5616             and last_char.char > 0x0C99 then
5617             quad = font.getfont(last_char.font).size
5618             for lg, rg in pairs(sea_ranges) do
5619                 if last_char.char > rg[1] and last_char.char < rg[2] then
5620                     lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril
5621                     local intraspace = Babel.intraspaces[lg]
5622                     local intrapenalty = Babel.intrapenalties[lg]
5623                     local n
5624                     if intrapenalty ~= 0 then
5625                         n = node.new(14, 0)      ^% penalty
5626                         n.penalty = intrapenalty
5627                         node.insert_before(head, item, n)
5628                     end
5629                     n = node.new(12, 13)      ^% (glue, spaceskip)
5630                     node.setglue(n, intraspace.b * quad,
5631                                 intraspace.p * quad,
5632                                 intraspace.m * quad)
5633                     node.insert_before(head, item, n)
5634                     node.remove(head, item)
5635                 end
5636             end
5637         end
5638     end
5639 end
5640 }^^
5641 \bbl@lua hyphenate}

```

## 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.

```

5642 \catcode`\%=14
5643 \gdef\bbl@cjk intraspace{%
5644     \let\bbl@cjk intraspace\relax
5645     \directlua{
5646         Babel = Babel or {}
5647         require('babel-data-cjk.lua')
5648         Babel.cjk_enabled = true
5649         function Babel.cjk_linebreak(head)
5650             local GLYPH = node.id'glyph'
5651             local last_char = nil
5652             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5653             local last_class = nil
5654             local last_lang = nil
5655
5656             for item in node.traverse(head) do
5657                 if item.id == GLYPH then
5658
5659                     local lang = item.lang
5660
5661                     local LOCALE = node.get_attribute(item,
5662                                                         Babel.attr_locale)
5663                     local props = Babel.locale_props[LOCALE]

```

```

5664
5665     local class = Babel.cjk_class[item.char].c
5666
5667     if props.cjk_quotes and props.cjk_quotes[item.char] then
5668         class = props.cjk_quotes[item.char]
5669     end
5670
5671     if class == 'cp' then class = 'cl' end % ]] as CL
5672     if class == 'id' then class = 'I' end
5673
5674     local br = 0
5675     if class and last_class and Babel.cjk_breaks[last_class][class] then
5676         br = Babel.cjk_breaks[last_class][class]
5677     end
5678
5679     if br == 1 and props.linebreak == 'c' and
5680         lang ~= \the\l@nohyphenation\space and
5681         last_lang ~= \the\l@nohyphenation then
5682         local intrapenalty = props.intrapenalty
5683         if intrapenalty ~= 0 then
5684             local n = node.new(14, 0)      % penalty
5685             n.penalty = intrapenalty
5686             node.insert_before(head, item, n)
5687         end
5688         local intraspace = props.intraspace
5689         local n = node.new(12, 13)        % (glue, spaceskip)
5690         node.setglue(n, intraspace.b * quad,
5691             intraspace.p * quad,
5692             intraspace.m * quad)
5693         node.insert_before(head, item, n)
5694     end
5695
5696     if font.getfont(item.font) then
5697         quad = font.getfont(item.font).size
5698     end
5699     last_class = class
5700     last_lang = lang
5701     else % if penalty, glue or anything else
5702         last_class = nil
5703     end
5704 end
5705 lang.hyphenate(head)
5706 end
5707 }%
5708 \bbl@lua hyphenate}
5709 \gdef\bbl@lua hyphenate{%
5710 \let\bbl@lua hyphenate\relax
5711 \directlua{
5712     luatexbase.add_to_callback('hyphenate',
5713     function (head, tail)
5714         if Babel.linebreaking.before then
5715             for k, func in ipairs(Babel.linebreaking.before) do
5716                 func(head)
5717             end
5718         end
5719         if Babel.cjk_enabled then
5720             Babel.cjk_linebreak(head)
5721         end
5722         lang.hyphenate(head)
5723         if Babel.linebreaking.after then
5724             for k, func in ipairs(Babel.linebreaking.after) do
5725                 func(head)
5726             end

```

```

5727     end
5728     if Babel.sea_enabled then
5729         Babel.sea_disc_to_space(head)
5730     end
5731 end,
5732 'Babel.hyphenate')
5733 }
5734 }
5735 \endgroup
5736 \def\bbl@provide@intraspace{%
5737   \bbl@ifunset{\bbl@intsp@{language}}{}%
5738   {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5739     \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}}%
5740   \ifin@           % cjk
5741     \bbl@cjk@intraspace
5742     \directlua{
5743       Babel = Babel or {}
5744       Babel.locale_props = Babel.locale_props or {}
5745       Babel.locale_props[\the\localeid].linebreak = 'c'
5746     }%
5747     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5748     \ifx\bbl@KVP@intrapenalty\@nnil
5749       \bbl@intrapenalty0\@{}
5750     \fi
5751   \else           % sea
5752     \bbl@sea@intraspace
5753     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5754     \directlua{
5755       Babel = Babel or {}
5756       Babel.sea_ranges = Babel.sea_ranges or {}
5757       Babel.set_chranges('\bbl@cl{sbc}',
5758         '\bbl@cl{chrng}')
5759     }%
5760     \ifx\bbl@KVP@intrapenalty\@nnil
5761       \bbl@intrapenalty0\@{}
5762     \fi
5763   \fi
5764   \fi
5765   \ifx\bbl@KVP@intrapenalty\@nnil\else
5766     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@{}
5767   \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-

```

5768 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5769 \def\bblar@chars{%
5770   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5771   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5772   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5773 \def\bblar@elongated{%
5774   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5775   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5776   0649,064A}
5777 \begin{group}
5778   \catcode`_ = 11 \catcode`\_ = 11
5779   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5780 \end{group}
5781 \gdef\bblar@arabicjust{% TODO. Allow for several locales.
5782   \let\bblar@arabicjust\relax
5783   \newattribute\bblar@kashida
5784   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%

```



```

5785 \bblar@kashida=\z@
5786 \bbl@patchfont{\bbl@parsejalt}}%
5787 \directlua{
5788   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5789   Babel.arabic.elong_map[\the\localeid] = {}
5790   luatexbase.add_to_callback('post_linebreak_filter',
5791     Babel.arabic.justify, 'Babel.arabic.justify')
5792   luatexbase.add_to_callback('hpack_filter',
5793     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5794 }%

```

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

```

5795 \def\bblar@fetchjalt#1#2#3#4{%
5796   \bbl@exp{\bbl@foreach{#1}}{%
5797     \bbl@ifunset\bblar@JE@##1{%
5798       {\setbox\z@\hbox{\texdir TRT ^^^200d\char"##1#2}}%
5799       {\setbox\z@\hbox{\texdir TRT ^^^200d\char"@nameuse\bblar@JE@##1#2}}%
5800     \directlua{%
5801       local last = nil
5802       for item in node.traverse(tex.box[0].head) do
5803         if item.id == node.id'glyph' and item.char > 0x600 and
5804           not (item.char == 0x200D) then
5805           last = item
5806         end
5807       end
5808       Babel.arabic.#3['##1#4'] = last.char
5809     }}

```

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

```

5810 \gdef\bbl@parsejalt{%
5811   \ifx\addfontfeature\undefined\else
5812     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5813     \ifin@
5814       \directlua{%
5815         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5816           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5817           tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5818         end
5819       }%
5820     \fi
5821   \fi}
5822 \gdef\bbl@parsejalti{%
5823   \begingroup
5824     \let\bbl@parsejalt\relax % To avoid infinite loop
5825     \edef\bbl@tempb{\fontid\font}%
5826     \bblar@nofswarn
5827     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5828     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5829     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5830     \addfontfeature{RawFeature+=jalt}%
5831     % \@namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5832     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5833     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5834     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5835     \directlua{%
5836       for k, v in pairs(Babel.arabic.from) do
5837         if Babel.arabic.dest[k] and
5838           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5839           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5840             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5841         end
5842       end
5843     }%

```

```
5844 \endgroup}
```

The actual justification (inspired by CHICKENIZE).

```
5845 \begingroup
5846 \catcode`#=11
5847 \catcode`~ =11
5848 \directlua{
5849
5850 Babel.arabic = Babel.arabic or {}
5851 Babel.arabic.from = {}
5852 Babel.arabic.dest = {}
5853 Babel.arabic.justify_factor = 0.95
5854 Babel.arabic.justify_enabled = true
5855 Babel.arabic.kashida_limit = -1
5856
5857 function Babel.arabic.justify(head)
5858   if not Babel.arabic.justify_enabled then return head end
5859   for line in node.traverse_id(node.id'hlist', head) do
5860     Babel.arabic.justify_hlist(head, line)
5861   end
5862   return head
5863 end
5864
5865 function Babel.arabic.justify_hbox(head, gc, size, pack)
5866   local has_inf = false
5867   if Babel.arabic.justify_enabled and pack == 'exactly' then
5868     for n in node.traverse_id(12, head) do
5869       if n.stretch_order > 0 then has_inf = true end
5870     end
5871     if not has_inf then
5872       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5873     end
5874   end
5875   return head
5876 end
5877
5878 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5879   local d, new
5880   local k_list, k_item, pos_inline
5881   local width, width_new, full, k_curr, wt_pos, goal, shift
5882   local subst_done = false
5883   local elong_map = Babel.arabic.elong_map
5884   local cnt
5885   local last_line
5886   local GLYPH = node.id'glyph'
5887   local KASHIDA = Babel.attr_kashida
5888   local LOCALE = Babel.attr_locale
5889
5890   if line == nil then
5891     line = {}
5892     line.glue_sign = 1
5893     line.glue_order = 0
5894     line.head = head
5895     line.shift = 0
5896     line.width = size
5897   end
5898
5899   % Exclude last line. todo. But-- it discards one-word lines, too!
5900   % ? Look for glue = 12:15
5901   if (line.glue_sign == 1 and line.glue_order == 0) then
5902     elongs = {} % Stores elongated candidates of each line
5903     k_list = {} % And all letters with kashida
5904     pos_inline = 0 % Not yet used
```

```

5905
5906 for n in node.traverse_id(GLYPH, line.head) do
5907     pos_inline = pos_inline + 1 % To find where it is. Not used.
5908
5909     % Elongated glyphs
5910     if elong_map then
5911         local locale = node.get_attribute(n, LOCALE)
5912         if elong_map[locale] and elong_map[locale][n.font] and
5913             elong_map[locale][n.font][n.char] then
5914             table.insert(elongs, {node = n, locale = locale} )
5915             node.set_attribute(n.prev, KASHIDA, 0)
5916         end
5917     end
5918
5919     % Tatwil
5920     if Babel.kashida_wts then
5921         local k_wt = node.get_attribute(n, KASHIDA)
5922         if k_wt > 0 then % todo. parameter for multi inserts
5923             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5924         end
5925     end
5926
5927 end % of node.traverse_id
5928
5929 if #elongs == 0 and #k_list == 0 then goto next_line end
5930 full = line.width
5931 shift = line.shift
5932 goal = full * Babel.arabic.justify_factor % A bit crude
5933 width = node.dimensions(line.head) % The 'natural' width
5934
5935 % == Elongated ==
5936 % Original idea taken from 'chickenize'
5937 while (#elongs > 0 and width < goal) do
5938     subst_done = true
5939     local x = #elongs
5940     local curr = elongs[x].node
5941     local oldchar = curr.char
5942     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5943     width = node.dimensions(line.head) % Check if the line is too wide
5944     % Substitute back if the line would be too wide and break:
5945     if width > goal then
5946         curr.char = oldchar
5947         break
5948     end
5949     % If continue, pop the just substituted node from the list:
5950     table.remove(elongs, x)
5951 end
5952
5953 % == Tatwil ==
5954 if #k_list == 0 then goto next_line end
5955
5956 width = node.dimensions(line.head) % The 'natural' width
5957 k_curr = #k_list % Traverse backwards, from the end
5958 wt_pos = 1
5959
5960 while width < goal do
5961     subst_done = true
5962     k_item = k_list[k_curr].node
5963     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5964         d = node.copy(k_item)
5965         d.char = 0x0640
5966         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5967         d.xoffset = 0

```

```

5968     line.head, new = node.insert_after(line.head, k_item, d)
5969     width_new = node.dimensions(line.head)
5970     if width > goal or width == width_new then
5971         node.remove(line.head, new) % Better compute before
5972         break
5973     end
5974     if Babel.fix_diacr then
5975         Babel.fix_diacr(k_item.next)
5976     end
5977     width = width_new
5978 end
5979 if k_curr == 1 then
5980     k_curr = #k_list
5981     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5982 else
5983     k_curr = k_curr - 1
5984 end
5985 end
5986
5987 % Limit the number of tatweel by removing them. Not very efficient,
5988 % but it does the job in a quite predictable way.
5989 if Babel.arabic.kashida_limit > -1 then
5990     cnt = 0
5991     for n in node.traverse_id(GLYPH, line.head) do
5992         if n.char == 0x0640 then
5993             cnt = cnt + 1
5994             if cnt > Babel.arabic.kashida_limit then
5995                 node.remove(line.head, n)
5996             end
5997         else
5998             cnt = 0
5999         end
6000     end
6001 end
6002
6003 ::next_line::
6004
6005 % Must take into account marks and ins, see luatex manual.
6006 % Have to be executed only if there are changes. Investigate
6007 % what's going on exactly.
6008 if subst_done and not gc then
6009     d = node.hpack(line.head, full, 'exactly')
6010     d.shift = shift
6011     node.insert_before(head, line, d)
6012     node.remove(head, line)
6013 end
6014 end % if process line
6015 end
6016 }
6017 \endgroup
6018 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

## 10.7 Common stuff

```

6019 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
6020 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
6021 \DisableBabelHook{babel-fontspec}
6022 <<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.

```

6023 % TODO - to a lua file
6024 \directlua{
6025 Babel.script_blocks = {
6026   ['dflt'] = {},
6027   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6028               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6029   ['Armn'] = {{0x0530, 0x058F}},
6030   ['Beng'] = {{0x0980, 0x09FF}},
6031   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6032   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6033   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6034               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6035   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6036   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6037               {0xAB00, 0xAB2F}},
6038   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6039   % Don't follow strictly Unicode, which places some Coptic letters in
6040   % the 'Greek and Coptic' block
6041   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6042   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6043               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6044               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6045               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6046               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6047               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6048   ['Hebr'] = {{0x0590, 0x05FF}},
6049   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6050               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6051   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6052   ['Knda'] = {{0x0C80, 0x0CFF}},
6053   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6054               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6055               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6056   ['Lao'] = {{0x0E80, 0x0EFF}},
6057   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6058               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6059               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6060   ['Mahj'] = {{0x11150, 0x1117F}},
6061   ['Mlym'] = {{0x0D00, 0x0D7F}},
6062   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6063   ['Orya'] = {{0x0B00, 0x0B7F}},
6064   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6065   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6066   ['Taml'] = {{0x0B80, 0x0BFF}},
6067   ['Telu'] = {{0x0C00, 0x0C7F}},
6068   ['Tfng'] = {{0x2D30, 0x2D7F}},
6069   ['Thai'] = {{0x0E00, 0x0E7F}},
6070   ['Tibt'] = {{0x0F00, 0x0FFF}},
6071   ['Vaii'] = {{0xA500, 0xA63F}},
6072   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6073 }
6074
6075 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyril
6076 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6077 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6078
6079 function Babel.locale_map(head)
6080   if not Babel.locale_mapped then return head end

```

```

6081
6082 local LOCALE = Babel.attr_locale
6083 local GLYPH = node.id('glyph')
6084 local inmath = false
6085 local toloc_save
6086 for item in node.traverse(head) do
6087   local toloc
6088   if not inmath and item.id == GLYPH then
6089     % Optimization: build a table with the chars found
6090     if Babel.chr_to_loc[item.char] then
6091       toloc = Babel.chr_to_loc[item.char]
6092     else
6093       for lc, maps in pairs(Babel.loc_to_scr) do
6094         for _, rg in pairs(maps) do
6095           if item.char >= rg[1] and item.char <= rg[2] then
6096             Babel.chr_to_loc[item.char] = lc
6097             toloc = lc
6098             break
6099           end
6100         end
6101       end
6102       % Treat composite chars in a different fashion, because they
6103       % 'inherit' the previous locale.
6104       if (item.char >= 0x0300 and item.char <= 0x036F) or
6105          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6106          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6107         Babel.chr_to_loc[item.char] = -2000
6108         toloc = -2000
6109       end
6110       if not toloc then
6111         Babel.chr_to_loc[item.char] = -1000
6112       end
6113     end
6114     if toloc == -2000 then
6115       toloc = toloc_save
6116     elseif toloc == -1000 then
6117       toloc = nil
6118     end
6119     if toloc and Babel.locale_props[toloc] and
6120        Babel.locale_props[toloc].letters and
6121        tex.getcatcode(item.char) \string~= 11 then
6122       toloc = nil
6123     end
6124     if toloc and Babel.locale_props[toloc].script
6125        and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6126        and Babel.locale_props[toloc].script ==
6127        Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6128       toloc = nil
6129     end
6130     if toloc then
6131       if Babel.locale_props[toloc].lg then
6132         item.lang = Babel.locale_props[toloc].lg
6133         node.set_attribute(item, LOCALE, toloc)
6134       end
6135       if Babel.locale_props[toloc]['/'..item.font] then
6136         item.font = Babel.locale_props[toloc]['/'..item.font]
6137       end
6138     end
6139     toloc_save = toloc
6140   elseif not inmath and item.id == 7 then % Apply recursively
6141     item.replace = item.replace and Babel.locale_map(item.replace)
6142     item.pre      = item.pre and Babel.locale_map(item.pre)
6143     item.post     = item.post and Babel.locale_map(item.post)

```

```

6144 elseif item.id == node.id'math' then
6145   inmath = (item.subtype == 0)
6146 end
6147 end
6148 return head
6149 end
6150 }

```

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

```

6151 \newcommand\babelcharproperty[1]{%
6152   \count@=#1\relax
6153   \ifvmode
6154     \expandafter\bbl@chprop
6155   \else
6156     \bbl@error{charproperty-only-vertical}{}{}{}%
6157   \fi}
6158 \newcommand\bbl@chprop[3][\the\count@]{%
6159   \@tempcnta=#1\relax
6160   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6161   {\bbl@error{unknown-char-property}{}{#2}}{}%
6162   {}%
6163   \loop
6164     \bbl@cs{chprop@#2}{#3}%
6165     \ifnum\count@<\@tempcnta
6166       \advance\count@ \@ne
6167     \repeat}
6168 \def\bbl@chprop@direction#1{%
6169   \directlua{
6170     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6171     Babel.characters[\the\count@]['d'] = '#1'
6172   }}
6173 \let\bbl@chprop@bc\bbl@chprop@direction
6174 \def\bbl@chprop@mirror#1{%
6175   \directlua{
6176     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6177     Babel.characters[\the\count@]['m'] = '\number#1'
6178   }}
6179 \let\bbl@chprop@bmg\bbl@chprop@mirror
6180 \def\bbl@chprop@linebreak#1{%
6181   \directlua{
6182     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6183     Babel.cjk_characters[\the\count@]['c'] = '#1'
6184   }}
6185 \let\bbl@chprop@lb\bbl@chprop@linebreak
6186 \def\bbl@chprop@locale#1{%
6187   \directlua{
6188     Babel.chr_to_loc = Babel.chr_to_loc or {}
6189     Babel.chr_to_loc[\the\count@] =
6190       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6191   }}

```

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.

```

6192 \directlua{
6193   Babel.nohyphenation = \the\l@nohyphenation
6194 }

```

Now the  $\TeX$  high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the `{n}` syntax. For example, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not

dissimilar to lua load – save the code as string in a TeX macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

```

6195 \begingroup
6196 \catcode`\~ = 12
6197 \catcode`\% = 12
6198 \catcode`\& = 14
6199 \catcode`\| = 12
6200 \gdef\babelprehyphenation{%&
6201   \@ifnextchar[{\babel@settransform{0}}{\babel@settransform{0}}{}}
6202 \gdef\babelposthyphenation{%&
6203   \@ifnextchar[{\babel@settransform{1}}{\babel@settransform{1}}{}}
6204 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6205   \ifcase#1
6206     \bbl@activateprehyphen
6207   \or
6208     \bbl@activateposthyphen
6209   \fi
6210 \begingroup
6211   \def\babeltempa{\bbl@add@list\babeltempb}%&
6212   \let\babeltempb@empty
6213   \def\bbl@tempa{#5}%&
6214   \bbl@replace\bbl@tempa{,}{ ,}%& TODO. Ugly trick to preserve {}
6215   \expandafter\bbl@foreach\expandafter{\babel@tempa}{%&
6216     \bbl@ifsamestring{##1}{remove}%&
6217     {\bbl@add@list\babeltempb{nil}}}%&
6218     {\directlua{
6219       local rep = {[##1]=}
6220       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6221       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6222       rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6223       if #1 == 0 or #1 == 2 then
6224         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6225           'space = { ' .. '%2, %3, %4' .. ' }')
6226         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6227           'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6228         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6229       else
6230         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6231         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6232         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6233       end
6234       tex.print([[string\babeltempa{}} .. rep .. [{}]])
6235     }]}%&
6236   \bbl@foreach\babeltempb{%&
6237     \bbl@forkv{##1}{%&
6238       \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,&
6239         no,post,penalty,kashida,space,spacefactor,}%&
6240       \ifin@else
6241         \bbl@error{bad-transform-option}{###1}{}%&
6242       \fi}%&
6243   \let\bbl@kv@attribute\relax
6244   \let\bbl@kv@label\relax
6245   \let\bbl@kv@fonts@empty
6246   \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}%&
6247   \ifx\bbl@kv@fonts@empty\else\bbl@settransform\fi
6248   \ifx\bbl@kv@attribute\relax
6249     \ifx\bbl@kv@label\relax\else
6250       \bbl@exp{\bbl@trim\def\bbl@kv@fonts{\bbl@kv@fonts}}%&
6251       \bbl@replace\bbl@kv@fonts{ }{ ,}%&
6252       \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}%&
6253       \count@\z@
6254       \def\bbl@elt##1##2##3{%&

```



```

6255     \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6256     {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6257     {\count@\@ne}&%
6258     {\bbl@error{font-conflict-transforms}{}}{}}}&%
6259     {}}&%
6260     \bbl@transfont@list
6261     \ifnum\count@=\z@
6262     \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6263     {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6264     \fi
6265     \bbl@ifunset{\bbl@kv@attribute}&%
6266     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6267     {}&%
6268     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6269     \fi
6270     \else
6271     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6272     \fi
6273     \directlua{
6274     local lbkr = Babel.linebreaking.replacements[#1]
6275     local u = unicode.utf8
6276     local id, attr, label
6277     if #1 == 0 then
6278     id = \the\csname bbl@id@#3\endcsname\space
6279     else
6280     id = \the\csname l@#3\endcsname\space
6281     end
6282     \ifx\bbl@kv@attribute\relax
6283     attr = -1
6284     \else
6285     attr = luatexbase.registernumber'\bbl@kv@attribute'
6286     \fi
6287     \ifx\bbl@kv@label\relax\else &% Same refs:
6288     label = [==[\bbl@kv@label]==]
6289     \fi
6290     &% Convert pattern:
6291     local patt = string.gsub([==[#4]==], '%s', '')
6292     if #1 == 0 then
6293     patt = string.gsub(patt, '|', ' ')
6294     end
6295     if not u.find(patt, '()', nil, true) then
6296     patt = '()' .. patt .. '()'
6297     end
6298     if #1 == 1 then
6299     patt = string.gsub(patt, '%(%)%^', '^()')
6300     patt = string.gsub(patt, '%$(%)', '()$')
6301     end
6302     patt = u.gsub(patt, '{(.)}',
6303     function (n)
6304     return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6305     end)
6306     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6307     function (n)
6308     return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6309     end)
6310     lbkr[id] = lbkr[id] or {}
6311     table.insert(lbkr[id],
6312     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6313     }&%
6314     \endgroup}
6315 \endgroup
6316 \let\bbl@transfont@list\@empty
6317 \def\bbl@settransfont{%

```

```

6318 \global\let\bbl@settransform\relax % Execute only once
6319 \gdef\bbl@transform{%
6320   \def\bbl@elt####1####2####3{%
6321     \bbl@ifblank{####3}%
6322     {\count@tw@}% Do nothing if no fonts
6323     {\count@z@
6324       \bbl@vforeach{####3}%
6325       \def\bbl@tempd{#####1}%
6326       \edef\bbl@tempe{\bbl@transform/\f@series/\f@shape}%
6327       \ifx\bbl@tempd\bbl@tempe
6328         \count@one
6329       \else\ifx\bbl@tempd\bbl@transform
6330         \count@one
6331       \fi\fi}%
6332     \ifcase\count@
6333       \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
6334     \or
6335       \bbl@csarg\setattribute{ATR####2@####1@####3}\@one
6336     \fi}}%
6337   \bbl@transform@list}%
6338 \AddToHook{selectfont}{\bbl@transform}% Hooks are global.
6339 \gdef\bbl@transform{-unknown-}%
6340 \bbl@foreach\bbl@font@fams{%
6341   \AddToHook{##1family}{\def\bbl@transform{##1}%
6342   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6343   {\xdef\bbl@transform{##1}}}%
6344   {}}
6345 \DeclareRobustCommand\enablelocaletransform[1]{%
6346   \bbl@ifunset{\bbl@ATR@#1@\language @}%
6347   {\bbl@error{transform-not-available}{#1}{}}}%
6348   {\bbl@csarg\setattribute{ATR@#1@\language @}\@one}}
6349 \DeclareRobustCommand\disablelocaletransform[1]{%
6350   \bbl@ifunset{\bbl@ATR@#1@\language @}%
6351   {\bbl@error{transform-not-available-b}{#1}{}}}%
6352   {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}%
6353 \def\bbl@activateposthyphen{%
6354   \let\bbl@activateposthyphen\relax
6355   \directlua{
6356     require('babel-transforms.lua')
6357     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6358   }}
6359 \def\bbl@activateprehyphen{%
6360   \let\bbl@activateprehyphen\relax
6361   \directlua{
6362     require('babel-transforms.lua')
6363     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6364   }}

```

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.

```

6365 \newcommand\localeprehyphenation[1]{%
6366   \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 luaoftload is applied, which is loaded by default by  $\LaTeX$ . Just in case, consider the possibility it has not been loaded.

```

6367 \def\bbl@activate@preotf{%
6368   \let\bbl@activate@preotf\relax % only once

```

```

6369 \directlua{
6370   Babel = Babel or {}
6371   %
6372   function Babel.pre_otfload_v(head)
6373     if Babel.numbers and Babel.digits_mapped then
6374       head = Babel.numbers(head)
6375     end
6376     if Babel.bidi_enabled then
6377       head = Babel.bidi(head, false, dir)
6378     end
6379     return head
6380   end
6381   %
6382   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6383     if Babel.numbers and Babel.digits_mapped then
6384       head = Babel.numbers(head)
6385     end
6386     if Babel.bidi_enabled then
6387       head = Babel.bidi(head, false, dir)
6388     end
6389     return head
6390   end
6391   %
6392   luatexbase.add_to_callback('pre_linebreak_filter',
6393     Babel.pre_otfload_v,
6394     'Babel.pre_otfload_v',
6395     luatexbase.priority_in_callback('pre_linebreak_filter',
6396       'luaotfload.node_processor') or nil)
6397   %
6398   luatexbase.add_to_callback('hpack_filter',
6399     Babel.pre_otfload_h,
6400     'Babel.pre_otfload_h',
6401     luatexbase.priority_in_callback('hpack_filter',
6402       'luaotfload.node_processor') or nil)
6403 }

```

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=`.

```

6404 \breakafterdirmode=1
6405 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6406   \let\bbl@beforeforeign\leavevmode
6407   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6408   \RequirePackage{luatexbase}
6409   \bbl@activate@preotf
6410   \directlua{
6411     require('babel-data-bidi.lua')
6412     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6413       require('babel-bidi-basic.lua')
6414     \or
6415       require('babel-bidi-basic-r.lua')
6416     \fi}
6417   \newattribute\bbl@attr@dir
6418   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6419   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6420 \fi
6421 \chardef\bbl@thetextdir\z@
6422 \chardef\bbl@thepardir\z@
6423 \def\bbl@getluadir#1{%
6424   \directlua{
6425     if tex.#ldir == 'TLT' then
6426       tex.sprint('0')
6427     elseif tex.#ldir == 'TRT' then

```

```

6428     tex.sprint('l')
6429   end}}
6430 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texdir.. 3=0 lr/1 r\
6431   \ifcase#3\relax
6432     \ifcase\bbl@getluadir{#1}\relax\else
6433       #2 TLT\relax
6434     \fi
6435   \else
6436     \ifcase\bbl@getluadir{#1}\relax
6437       #2 TRT\relax
6438     \fi
6439   \fi}
6440 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6441 \def\bbl@thedir{0}
6442 \def\bbl@texdir#1{%
6443   \bbl@setluadir{text}\texdir{#1}%
6444   \chardef\bbl@thetexdir#1\relax
6445   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6446   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6447 \def\bbl@pardir#1{% Used twice
6448   \bbl@setluadir{par}\pardir{#1}%
6449   \chardef\bbl@thepardir#1\relax}
6450 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6451 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6452 \def\bbl@dirparastext{\pardir\the\texdir\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.

```

6453 \ifnum\bbl@bidimode>\z@ % Any bidi=
6454   \def\bbl@insidemath{0}%
6455   \def\bbl@everymath{\def\bbl@insidemath{1}}
6456   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6457   \frozen@everymath\expandafter{%
6458     \expandafter\bbl@everymath\the\frozen@everymath}
6459   \frozen@everydisplay\expandafter{%
6460     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6461   \AtBeginDocument{
6462     \directlua{
6463       function Babel.math_box_dir(head)
6464         if not (token.get_macro('bbl@insidemath') == '0') then
6465           if Babel.hlist_has_bidi(head) then
6466             local d = node.new(node.id'dir')
6467             d.dir = '+TRT'
6468             node.insert_before(head, node.has_glyph(head), d)
6469             for item in node.traverse(head) do
6470               node.set_attribute(item,
6471                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6472             end
6473           end
6474         end
6475         return head
6476       end
6477       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6478         "Babel.math_box_dir", 0)
6479     }}%
6480 \fi

```

## 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, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```

6481 \bbl@trace{Redefinitions for bidi layout}
6482 %
6483 <<(*More package options)>> ≡
6484 \chardef\bbl@eqnpos\z@
6485 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6486 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6487 <</More package options>>
6488 %
6489 \ifnum\bbl@bidimode>\z@ % Any bidi=
6490   \matheqdirmode\@ne % A luatex primitive
6491   \let\bbl@eqnodir\relax
6492   \def\bbl@eqdel{()}
6493   \def\bbl@eqnum{%
6494     {\normalfont\normalcolor
6495       \expandafter\@firstoftwo\bbl@eqdel
6496       \theequation
6497       \expandafter\@secondoftwo\bbl@eqdel}}
6498   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6499   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6500   \def\bbl@eqno@flip#1{%
6501     \ifdim\predisplaysize=-\maxdimen
6502       \eqno
6503       \hb@xt@.01pt{%
6504         \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6505     \else
6506       \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6507     \fi
6508     \bbl@exp{\def\\@currentlabel{[\bbl@upset]}}
6509   \def\bbl@leqno@flip#1{%
6510     \ifdim\predisplaysize=-\maxdimen
6511       \leqno
6512       \hb@xt@.01pt{%
6513         \hss\hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6514     \else
6515       \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6516     \fi
6517     \bbl@exp{\def\\@currentlabel{[\bbl@upset]}}
6518   \AtBeginDocument{%
6519     \ifx\bbl@noamsmath\relax\else
6520     \ifx\maketag@@@undefined % Normal equation, eqnarray
6521       \AddToHook{env/equation/begin}{%
6522         \ifnum\bbl@thetextdir>\z@
6523           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6524           \let\@eqnnum\bbl@eqnum
6525           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6526           \chardef\bbl@thetextdir\z@
6527           \bbl@add\normalfont{\bbl@eqnodir}%
6528           \ifcase\bbl@eqnpos

```

```

6529         \let\bbl@puteqno\bbl@eqno@flip
6530     \or
6531         \let\bbl@puteqno\bbl@leqno@flip
6532     \fi
6533 \fi}%
6534 \ifnum\bbl@eqnpos=\tw@%else
6535     \def\endequation{\bbl@puteqno{\@eqnnum}$\@ignoretrue}%
6536 \fi
6537 \AddToHook{env/eqnarray/begin}{%
6538     \ifnum\bbl@thetextdir>\z@
6539         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6540         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6541         \chardef\bbl@thetextdir\z@
6542         \bbl@add\normalfont{\bbl@eqnodir}%
6543         \ifnum\bbl@eqnpos=\@ne
6544             \def\@eqnnum{%
6545                 \setbox\z@\hbox{\bbl@eqnum}%
6546                 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6547         \else
6548             \let\@eqnnum\bbl@eqnum
6549         \fi
6550     \fi}
6551 % Hack. YA luatex bug?:
6552 \expandafter\bbl@sreplace\csname] \endcsname{$$}\@eqno\kern.001pt$}%
6553 \else % amstex
6554     \bbl@exp{% Hack to hide maybe undefined conditionals:
6555         \chardef\bbl@eqnpos=0%
6556         \<iftagsleft>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6557     \ifnum\bbl@eqnpos=\@ne
6558         \let\bbl@ams@lap\hbox
6559     \else
6560         \let\bbl@ams@lap\llap
6561     \fi
6562     \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6563     \bbl@sreplace\intertext@{\normalbaselines}%
6564     {\normalbaselines
6565         \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6566     \ExplSyntaxOff
6567     \def\bbl@ams@tagbox#1#2#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6568     \ifx\bbl@ams@lap\hbox % leqno
6569         \def\bbl@ams@flip#1{%
6570             \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6571     \else % eqno
6572         \def\bbl@ams@flip#1{%
6573             \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6574     \fi
6575     \def\bbl@ams@preset#1{%
6576         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6577         \ifnum\bbl@thetextdir>\z@
6578             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6579             \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6580             \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6581         \fi}%
6582     \ifnum\bbl@eqnpos=\tw@%else
6583         \def\bbl@ams@equation{%
6584             \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6585             \ifnum\bbl@thetextdir>\z@
6586                 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6587                 \chardef\bbl@thetextdir\z@
6588                 \bbl@add\normalfont{\bbl@eqnodir}%
6589                 \ifcase\bbl@eqnpos
6590                     \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6591                 \or

```

```

6592         \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6593     \fi
6594 \fi}%
6595     \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6596     \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6597 \fi
6598 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6599 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6600 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6601 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6602 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6603 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6604 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6605 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6606 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6607 % Hackish, for proper alignment. Don't ask me why it works!:
6608 \bbl@exp{% Avoid a 'visible' conditional
6609     \\ \AddToHook{env/align*/end}{\<iftag@>\<else>\\ \tag*{} \<fi>}%
6610     \\ \AddToHook{env/alignat*/end}{\<iftag@>\<else>\\ \tag*{} \<fi>}}%
6611 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6612 \AddToHook{env/split/before}{%
6613     \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6614     \ifnum\bbl@thetextdir>\z@
6615         \bbl@ifsamestring\@currentvir{equation}%
6616         {\ifx\bbl@ams@lap\hbox % leqno
6617             \def\bbl@ams@flip#1{%
6618                 \hbox to 0.01pt{\hbox to\displaywidth{##1}\hss}\hss}}%
6619             \else
6620                 \def\bbl@ams@flip#1{%
6621                     \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{##1}}}}%
6622             \fi}%
6623         }%
6624     \fi}%
6625 \fi\fi}
6626 \fi
6627 \def\bbl@provide@extra#1{%
6628     % == Counters: mapdigits ==
6629     % Native digits
6630     \ifx\bbl@KVP@mapdigits\@nnil\else
6631         \bbl@ifunset{\bbl@dgnat@language}{%
6632             {\RequirePackage{luatexbase}}%
6633             \bbl@activate@preotf
6634             \directlua{
6635                 Babel = Babel or {} %%% -> presets in luababel
6636                 Babel.digits_mapped = true
6637                 Babel.digits = Babel.digits or {}
6638                 Babel.digits[\the\localeid] =
6639                     table.pack(string.utfvalue('\bbl@cldgnat'))
6640                 if not Babel.numbers then
6641                     function Babel.numbers(head)
6642                         local LOCALE = Babel.attr_locale
6643                         local GLYPH = node.id'glyph'
6644                         local inmath = false
6645                         for item in node.traverse(head) do
6646                             if not inmath and item.id == GLYPH then
6647                                 local temp = node.get_attribute(item, LOCALE)
6648                                 if Babel.digits[temp] then
6649                                     local chr = item.char
6650                                     if chr > 47 and chr < 58 then
6651                                         item.char = Babel.digits[temp][chr-47]
6652                                     end
6653                                 end
6654                                 elseif item.id == node.id'math' then

```

```

6655             inmath = (item.subtype == 0)
6656         end
6657     end
6658     return head
6659 end
6660 end
6661 }}%
6662 \fi
6663 % == transforms ==
6664 \ifx\bb@KVP@transforms\@nnil\else
6665   \def\bb@elt##1##2##3{%
6666     \in@{${transforms.}}{##1}%
6667     \ifin@
6668       \def\bb@tempa{##1}%
6669       \bb@replace\bb@tempa{transforms.}{}%
6670       \bb@carg\bb@transforms{babel\bb@tempa}{##2}{##3}%
6671     \fi}%
6672   \csname bbl@inidata@\language\endcsname
6673   \bb@release@transforms\relax % \relax closes the last item.
6674 \fi}
6675 % Start tabular here:
6676 \def\localerestoredirs{%
6677   \ifcase\bb@thetextdir
6678     \ifnum\textdirection=\z@\else\textdir TLT\fi
6679   \else
6680     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6681   \fi
6682   \ifcase\bb@thepardir
6683     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6684   \else
6685     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6686   \fi}
6687 \IfBabelLayout{tabular}%
6688   {\chardef\bb@tabular@mode\tw}% All RTL
6689   {\IfBabelLayout{notabular}%
6690     {\chardef\bb@tabular@mode\z}%
6691     {\chardef\bb@tabular@mode\@ne}}% Mixed, with LTR cols
6692 \ifnum\bb@bidimode>\@ne % Any lua bidi= except default=1
6693   \ifcase\bb@tabular@mode\or % 1
6694     \let\bb@parabefore\relax
6695     \AddToHook{para/before}{\bb@parabefore}
6696     \AtBeginDocument{%
6697       \bb@replace\@tabular{$}{$}%
6698       \def\bb@insidemath{0}%
6699       \def\bb@parabefore{\localerestoredirs}}%
6700     \ifnum\bb@tabular@mode=\@ne
6701       \bb@ifunset{\@tabclassz}{}%
6702       \bb@exp{% Hide conditionals
6703         \\bb@sreplace\\ \@tabclassz
6704         {\<ifcase>\\ \@chnum}%
6705         {\localerestoredirs\<ifcase>\\ \@chnum}}}%
6706       \ifpackageloaded{colortbl}%
6707         {\bb@sreplace\@classz
6708           {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6709         {\@ifpackageloaded{array}%
6710           {\bb@exp{% Hide conditionals
6711             \\bb@sreplace\\ \@classz
6712             {\<ifcase>\\ \@chnum}%
6713             {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6714             \\bb@sreplace\\ \@classz
6715             {\do@row@strut\<fi>{\do@row@strut\<fi>\egroup}}}%
6716           }}%
6717     \fi}%

```



```

6718 \or % 2
6719 \let\bbl@parabefore\relax
6720 \AddToHook{para/before}{\bbl@parabefore}%
6721 \AtBeginDocument{%
6722 \ifpackageloaded{colortbl}%
6723 {\bbl@replace\@tabular{$}{$}%
6724 \def\bbl@insidemath{0}%
6725 \def\bbl@parabefore{\localerestoredirs}}%
6726 \bbl@sreplace\@classz
6727 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6728 {}}%
6729 \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.

```

6730 \AtBeginDocument{%
6731 \ifpackageloaded{multicol}%
6732 {\toks@\expandafter{\multi@column@out}%
6733 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6734 {}}%
6735 \ifpackageloaded{paracol}%
6736 {\edef\pcol@output{%
6737 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6738 {}}%
6739 \fi
6740 \ifx\bbl@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. `\bbl@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`.

```

6741 \ifnum\bbl@bidimode>\z@ % Any bidi=
6742 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6743 \bbl@exp{%
6744 \def\\bbl@insidemath{0}%
6745 \mathdir\the\bodydir
6746 #1% Once entered in math, set boxes to restore values
6747 \<ifmmode>%
6748 \everyvbox{%
6749 \the\everyvbox
6750 \bodydir\the\bodydir
6751 \mathdir\the\mathdir
6752 \everyhbox{\the\everyhbox}%
6753 \everyvbox{\the\everyvbox}}%
6754 \everyhbox{%
6755 \the\everyhbox
6756 \bodydir\the\bodydir
6757 \mathdir\the\mathdir
6758 \everyhbox{\the\everyhbox}%
6759 \everyvbox{\the\everyvbox}}%
6760 \<fi>}}%
6761 \def\@hangfrom#1{%
6762 \setbox\@tempboxa\hbox{#1}}%
6763 \hangindent\wd\@tempboxa
6764 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6765 \shapemode@ne
6766 \fi
6767 \noindent\box\@tempboxa}
6768 \fi
6769 \IfBabelLayout{tabular}
6770 {\let\bbl@OL@tabular\@tabular
6771 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6772 \let\bbl@NL@tabular\@tabular

```

```

6773 \AtBeginDocument{%
6774   \ifx\bbL@NL@tabular\@tabular\else
6775     \bbL@exp{\in@{\bbL@nextfake}{\@tabular}}}%
6776   \ifin\else
6777     \bbL@replace\@tabular{$}{\bbL@nextfake$}%
6778   \fi
6779   \let\bbL@NL@tabular\@tabular
6780 \fi}}
6781 {}
6782 \IfBabelLayout{lists}
6783 {\let\bbL@OL@list\list
6784  \bbL@sreplace\list{\parshape}{\bbL@listparshape}%
6785  \let\bbL@NL@list\list
6786  \def\bbL@listparshape#1#2#3{%
6787    \parshape #1 #2 #3 %
6788    \ifnum\bbL@getluadir{page}=\bbL@getluadir{par}\else
6789      \shapemode\tw@
6790    \fi}}
6791 {}
6792 \IfBabelLayout{graphics}
6793 {\let\bbL@pictresetdir\relax
6794  \def\bbL@pictsetdir#1{%
6795    \ifcase\bbL@thetextdir
6796      \let\bbL@pictresetdir\relax
6797    \else
6798      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6799        \or\textdir TLT
6800        \else\bodydir TLT \textdir TLT
6801      \fi
6802      % \(\text|par)dir required in pgf:
6803      \def\bbL@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6804    \fi}%
6805  \AddToHook{env/picture/begin}{\bbL@pictsetdir\tw@}%
6806  \directlua{
6807    Babel.get_picture_dir = true
6808    Babel.picture_has_bidi = 0
6809    %
6810    function Babel.picture_dir (head)
6811      if not Babel.get_picture_dir then return head end
6812      if Babel.hlist_has_bidi(head) then
6813        Babel.picture_has_bidi = 1
6814      end
6815      return head
6816    end
6817    luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6818      "Babel.picture_dir")
6819  }%
6820 \AtBeginDocument{%
6821   \def\LS@rot{%
6822     \setbox\@outputbox\vbox{%
6823       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6824   \long\def\put(#1,#2)#3{%
6825     \@killglue
6826     % Try:
6827     \ifx\bbL@pictresetdir\relax
6828       \def\bbL@tempc{0}%
6829     \else
6830       \directlua{
6831         Babel.get_picture_dir = true
6832         Babel.picture_has_bidi = 0
6833       }%
6834       \setbox\z@\hb@xt@{z}{%
6835         \@defaultunitsset\@tempdimc{#1}\unitlength

```

```

6836      \kern\@tempdimc
6837      #3\hss}% TODO: #3 executed twice (below). That's bad.
6838      \edef\bbL@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
6839      \fi
6840      % Do:
6841      \@defaultunitsset\@tempdimc{#2}\unitlength
6842      \raise\@tempdimc\hbext@z@{%
6843      \@defaultunitsset\@tempdimc{#1}\unitlength
6844      \kern\@tempdimc
6845      {\ifnum\bbL@tempc>z@\bbL@pictresetdir\fi#3}\hss}%
6846      \ignorespaces}%
6847      \MakeRobust\put}%
6848      \AtBeginDocument
6849      {\AddToHook{cmd/diagbox@pict/before}{\let\bbL@pictsetdir@gobble}%
6850      \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6851      \AddToHook{env/pgfpicture/begin}{\bbL@pictsetdir@ne}%
6852      \bbL@add\pgfinterruptpicture{\bbL@pictresetdir}%
6853      \bbL@add\pgfsys@beginpicture{\bbL@pictsetdir\z@}%
6854      \fi
6855      \ifx\tikzpicture\@undefined\else
6856      \AddToHook{env/tikzpicture/begin}{\bbL@pictsetdir\tw@}%
6857      \bbL@add\tikz@atbegin@node{\bbL@pictresetdir}%
6858      \bbL@sreplace\tikz{\begingroup}{\begingroup\bbL@pictsetdir\tw@}%
6859      \fi
6860      \ifx\tcolorbox\@undefined\else
6861      \def\tcb@drawing@env@begin{%
6862      \csname tcb@before@tcb@split@state\endcsname
6863      \bbL@pictsetdir\tw@
6864      \begin{\kv tcb@graphenv}%
6865      \tcb@bbdraw%
6866      \tcb@apply@graph@patches
6867      }%
6868      \def\tcb@drawing@env@end{%
6869      \end{\kv tcb@graphenv}%
6870      \bbL@pictresetdir
6871      \csname tcb@after@tcb@split@state\endcsname
6872      }%
6873      \fi
6874      }}
6875      {}

```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

```

6876 \IfBabelLayout{counters*}%
6877 {\bbL@add\bbL@opt@layout{.counters.}%
6878 \directlua{
6879 \lua{texbase.add_to_callback("process_output_buffer",
6880 Babel.discard_sublr , "Babel.discard_sublr") }%
6881 }}
6882 \IfBabelLayout{counters}%
6883 {\let\bbL@OL@@textsuperscript\@textsuperscript
6884 \bbL@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6885 \let\bbL@latinarabic=\@arabic
6886 \let\bbL@OL@@arabic\@arabic
6887 \def\@arabic#1{\babelsublr{\bbL@latinarabic#1}}%
6888 \@ifpackagewith{babel}{bidi=default}%
6889 {\let\bbL@asciroman=\@roman
6890 \let\bbL@OL@@roman\@roman
6891 \def\@roman#1{\babelsublr{\ensureascii{\bbL@asciroman#1}}}%
6892 \let\bbL@asciiRoman=\@Roman
6893 \let\bbL@OL@@roman\@Roman
6894 \def\@Roman#1{\babelsublr{\ensureascii{\bbL@asciiRoman#1}}}%

```

```

6895 \let\bbl@0L@labelenumii\labelenumii
6896 \def\labelenumii{}\theenumii}%
6897 \let\bbl@0L@p@enumiii\p@enumiii
6898 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}
6899 <<Footnote changes>>
6900 \IfBabelLayout{footnotes}%
6901 {\let\bbl@0L@footnote\footnote
6902 \BabelFootnote\footnote\languagename{}\}\}%
6903 \BabelFootnote\localfootnote\languagename{}\}\}%
6904 \BabelFootnote\mainfootnote{}\}\}\}
6905 {}

```

Some  $\text{\LaTeX}$  macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```

6906 \IfBabelLayout{extras}%
6907 {\bbl@ncarg\let\bbl@0L@underline{underline }%
6908 \bbl@carg\bbl@sreplace{underline }%
6909 {\$@@underline}\bgroup\bbl@nextfake$@@underline}%
6910 \bbl@carg\bbl@sreplace{underline }%
6911 {\m@th$}\m@th$\egroup}%
6912 \let\bbl@0L@LaTeXe\LaTeXe
6913 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6914 \if b\expandafter\@car\@series\@nil\boldmath\fi
6915 \babelsublr}%
6916 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
6917 {}
6918 </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.

```

6919 <*transforms>
6920 Babel.linebreaking.replacements = {}
6921 Babel.linebreaking.replacements[0] = {} -- pre
6922 Babel.linebreaking.replacements[1] = {} -- post
6923
6924 -- Discretionaries contain strings as nodes
6925 function Babel.str_to_nodes(fn, matches, base)
6926 local n, head, last
6927 if fn == nil then return nil end
6928 for s in string.utfvalues(fn(matches)) do
6929 if base.id == 7 then
6930 base = base.replace
6931 end
6932 n = node.copy(base)
6933 n.char = s
6934 if not head then
6935 head = n
6936 else
6937 last.next = n
6938 end
6939 last = n
6940 end

```

```

6941 return head
6942 end
6943
6944 Babel.fetch_subtext = {}
6945
6946 Babel.ignore_pre_char = function(node)
6947 return (node.lang == Babel.nohyphenation)
6948 end
6949
6950 -- Merging both functions doesn't seem feasible, because there are too
6951 -- many differences.
6952 Babel.fetch_subtext[0] = function(head)
6953 local word_string = ''
6954 local word_nodes = {}
6955 local lang
6956 local item = head
6957 local inmath = false
6958
6959 while item do
6960
6961     if item.id == 11 then
6962         inmath = (item.subtype == 0)
6963     end
6964
6965     if inmath then
6966         -- pass
6967
6968     elseif item.id == 29 then
6969         local locale = node.get_attribute(item, Babel.attr_locale)
6970
6971         if lang == locale or lang == nil then
6972             lang = lang or locale
6973             if Babel.ignore_pre_char(item) then
6974                 word_string = word_string .. Babel.us_char
6975             else
6976                 word_string = word_string .. unicode.utf8.char(item.char)
6977             end
6978             word_nodes[#word_nodes+1] = item
6979         else
6980             break
6981         end
6982
6983     elseif item.id == 12 and item.subtype == 13 then
6984         word_string = word_string .. ' '
6985         word_nodes[#word_nodes+1] = item
6986
6987         -- Ignore leading unrecognized nodes, too.
6988     elseif word_string ~= '' then
6989         word_string = word_string .. Babel.us_char
6990         word_nodes[#word_nodes+1] = item -- Will be ignored
6991     end
6992
6993     item = item.next
6994 end
6995
6996 -- Here and above we remove some trailing chars but not the
6997 -- corresponding nodes. But they aren't accessed.
6998 if word_string:sub(-1) == ' ' then
6999     word_string = word_string:sub(1,-2)
7000 end
7001 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7002 return word_string, word_nodes, item, lang
7003 end

```

```

7004
7005 Babel.fetch_subtext[1] = function(head)
7006   local word_string = ''
7007   local word_nodes = {}
7008   local lang
7009   local item = head
7010   local inmath = false
7011
7012   while item do
7013
7014     if item.id == 11 then
7015       inmath = (item.subtype == 0)
7016     end
7017
7018     if inmath then
7019       -- pass
7020
7021     elseif item.id == 29 then
7022       if item.lang == lang or lang == nil then
7023         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7024           lang = lang or item.lang
7025           word_string = word_string .. unicode.utf8.char(item.char)
7026           word_nodes[#word_nodes+1] = item
7027         end
7028       else
7029         break
7030       end
7031
7032     elseif item.id == 7 and item.subtype == 2 then
7033       word_string = word_string .. '='
7034       word_nodes[#word_nodes+1] = item
7035
7036     elseif item.id == 7 and item.subtype == 3 then
7037       word_string = word_string .. '|'
7038       word_nodes[#word_nodes+1] = item
7039
7040     -- (1) Go to next word if nothing was found, and (2) implicitly
7041     -- remove leading USs.
7042     elseif word_string == '' then
7043       -- pass
7044
7045     -- This is the responsible for splitting by words.
7046     elseif (item.id == 12 and item.subtype == 13) then
7047       break
7048
7049     else
7050       word_string = word_string .. Babel.us_char
7051       word_nodes[#word_nodes+1] = item -- Will be ignored
7052     end
7053
7054     item = item.next
7055   end
7056
7057   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7058   return word_string, word_nodes, item, lang
7059 end
7060
7061 function Babel.pre_hyphenate_replace(head)
7062   Babel.hyphenate_replace(head, 0)
7063 end
7064
7065 function Babel.post_hyphenate_replace(head)
7066   Babel.hyphenate_replace(head, 1)

```

```

7067 end
7068
7069 Babel.us_char = string.char(31)
7070
7071 function Babel.hyphenate_replace(head, mode)
7072     local u = unicode.utf8
7073     local lbkr = Babel.linebreaking.replacements[mode]
7074
7075     local word_head = head
7076
7077     while true do -- for each subtext block
7078
7079         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7080
7081         if Babel.debug then
7082             print()
7083             print((mode == 0) and '@@@@<' or '@@@@>', w)
7084         end
7085
7086         if nw == nil and w == '' then break end
7087
7088         if not lang then goto next end
7089         if not lbkr[lang] then goto next end
7090
7091         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7092         -- loops are nested.
7093         for k=1, #lbkr[lang] do
7094             local p = lbkr[lang][k].pattern
7095             local r = lbkr[lang][k].replace
7096             local attr = lbkr[lang][k].attr or -1
7097
7098             if Babel.debug then
7099                 print('*****', p, mode)
7100             end
7101
7102             -- This variable is set in some cases below to the first *byte*
7103             -- after the match, either as found by u.match (faster) or the
7104             -- computed position based on sc if w has changed.
7105             local last_match = 0
7106             local step = 0
7107
7108             -- For every match.
7109             while true do
7110                 if Babel.debug then
7111                     print('====')
7112                 end
7113                 local new -- used when inserting and removing nodes
7114
7115                 local matches = { u.match(w, p, last_match) }
7116
7117                 if #matches < 2 then break end
7118
7119                 -- Get and remove empty captures (with ())'s, which return a
7120                 -- number with the position), and keep actual captures
7121                 -- (from (...)), if any, in matches.
7122                 local first = table.remove(matches, 1)
7123                 local last = table.remove(matches, #matches)
7124                 -- Non re-fetched substrings may contain \31, which separates
7125                 -- subsubstrings.
7126                 if string.find(w:sub(first, last-1), Babel.us_char) then break end
7127
7128                 local save_last = last -- with A()BC()D, points to D
7129

```

```

7130 -- Fix offsets, from bytes to unicode. Explained above.
7131 first = u.len(w:sub(1, first-1)) + 1
7132 last = u.len(w:sub(1, last-1)) -- now last points to C
7133
7134 -- This loop stores in a small table the nodes
7135 -- corresponding to the pattern. Used by 'data' to provide a
7136 -- predictable behavior with 'insert' (w_nodes is modified on
7137 -- the fly), and also access to 'remove'd nodes.
7138 local sc = first-1 -- Used below, too
7139 local data_nodes = {}
7140
7141 local enabled = true
7142 for q = 1, last-first+1 do
7143     data_nodes[q] = w_nodes[sc+q]
7144     if enabled
7145         and attr > -1
7146         and not node.has_attribute(data_nodes[q], attr)
7147     then
7148         enabled = false
7149     end
7150 end
7151
7152 -- This loop traverses the matched substring and takes the
7153 -- corresponding action stored in the replacement list.
7154 -- sc = the position in substr nodes / string
7155 -- rc = the replacement table index
7156 local rc = 0
7157
7158 while rc < last-first+1 do -- for each replacement
7159     if Babel.debug then
7160         print('.....', rc + 1)
7161     end
7162     sc = sc + 1
7163     rc = rc + 1
7164
7165     if Babel.debug then
7166         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7167         local ss = ''
7168         for itt in node.traverse(head) do
7169             if itt.id == 29 then
7170                 ss = ss .. unicode.utf8.char(itt.char)
7171             else
7172                 ss = ss .. '{' .. itt.id .. '}'
7173             end
7174         end
7175         print('*****', ss)
7176     end
7177
7178     local crep = r[rc]
7179     local item = w_nodes[sc]
7180     local item_base = item
7181     local placeholder = Babel.us_char
7182     local d
7183
7184     if crep and crep.data then
7185         item_base = data_nodes[crep.data]
7186     end
7187
7188     if crep then
7189         step = crep.step or 0
7190     end
7191 end
7192

```



```

7193     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7194         last_match = save_last    -- Optimization
7195         goto next
7196
7197     elseif crep == nil or crep.remove then
7198         node.remove(head, item)
7199         table.remove(w_nodes, sc)
7200         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7201         sc = sc - 1 -- Nothing has been inserted.
7202         last_match = utf8.offset(w, sc+1+step)
7203         goto next
7204
7205     elseif crep and crep.kashida then -- Experimental
7206         node.set_attribute(item,
7207             Babel.attr_kashida,
7208             crep.kashida)
7209         last_match = utf8.offset(w, sc+1+step)
7210         goto next
7211
7212     elseif crep and crep.string then
7213         local str = crep.string(matches)
7214         if str == '' then -- Gather with nil
7215             node.remove(head, item)
7216             table.remove(w_nodes, sc)
7217             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7218             sc = sc - 1 -- Nothing has been inserted.
7219         else
7220             local loop_first = true
7221             for s in string.utfvalues(str) do
7222                 d = node.copy(item_base)
7223                 d.char = s
7224                 if loop_first then
7225                     loop_first = false
7226                     head, new = node.insert_before(head, item, d)
7227                     if sc == 1 then
7228                         word_head = head
7229                     end
7230                     w_nodes[sc] = d
7231                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7232                 else
7233                     sc = sc + 1
7234                     head, new = node.insert_before(head, item, d)
7235                     table.insert(w_nodes, sc, new)
7236                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7237                 end
7238                 if Babel.debug then
7239                     print('.....', 'str')
7240                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7241                 end
7242             end -- for
7243             node.remove(head, item)
7244         end -- if ''
7245         last_match = utf8.offset(w, sc+1+step)
7246         goto next
7247
7248     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7249         d = node.new(7, 3) -- (disc, regular)
7250         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7251         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7252         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7253         d.attr = item_base.attr
7254         if crep.pre == nil then -- TeXbook p96
7255             d.penalty = crep.penalty or tex.hyphenpenalty

```

```

7256         else
7257             d.penalty = crep.penalty or tex.exhyphenpenalty
7258         end
7259         placeholder = '|'
7260         head, new = node.insert_before(head, item, d)
7261
7262     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7263         -- ERROR
7264
7265     elseif crep and crep.penalty then
7266         d = node.new(14, 0) -- (penalty, userpenalty)
7267         d.attr = item_base.attr
7268         d.penalty = crep.penalty
7269         head, new = node.insert_before(head, item, d)
7270
7271     elseif crep and crep.space then
7272         -- 655360 = 10 pt = 10 * 65536 sp
7273         d = node.new(12, 13) -- (glue, spaceskip)
7274         local quad = font.getfont(item_base.font).size or 655360
7275         node.setglue(d, crep.space[1] * quad,
7276                       crep.space[2] * quad,
7277                       crep.space[3] * quad)
7278         if mode == 0 then
7279             placeholder = ' '
7280         end
7281         head, new = node.insert_before(head, item, d)
7282
7283     elseif crep and crep.spacefactor then
7284         d = node.new(12, 13) -- (glue, spaceskip)
7285         local base_font = font.getfont(item_base.font)
7286         node.setglue(d,
7287                     crep.spacefactor[1] * base_font.parameters['space'],
7288                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
7289                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
7290         if mode == 0 then
7291             placeholder = ' '
7292         end
7293         head, new = node.insert_before(head, item, d)
7294
7295     elseif mode == 0 and crep and crep.space then
7296         -- ERROR
7297
7298     end -- ie replacement cases
7299
7300     -- Shared by disc, space and penalty.
7301     if sc == 1 then
7302         word_head = head
7303     end
7304     if crep.insert then
7305         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7306         table.insert(w_nodes, sc, new)
7307         last = last + 1
7308     else
7309         w_nodes[sc] = d
7310         node.remove(head, item)
7311         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7312     end
7313
7314     last_match = utf8.offset(w, sc+1+step)
7315
7316     ::next::
7317
7318 end -- for each replacement

```

```

7319
7320         if Babel.debug then
7321             print('.....', '/')
7322             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7323         end
7324     end
7325 end -- for match
7326
7327 end -- for patterns
7328
7329 ::next::
7330     word_head = nw
7331 end -- for substring
7332 return head
7333 end
7334
7335 -- This table stores capture maps, numbered consecutively
7336 Babel.capture_maps = {}
7337
7338 -- The following functions belong to the next macro
7339 function Babel.capture_func(key, cap)
7340     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[") .. "]"
7341     local cnt
7342     local u = unicode.utf8
7343     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7344     if cnt == 0 then
7345         ret = u.gsub(ret, '{{%x%x%x%x+}}',
7346             function (n)
7347                 return u.char(tonumber(n, 16))
7348             end)
7349     end
7350     ret = ret:gsub("%[%[%]]%.%", '')
7351     ret = ret:gsub("%.%.%[%[%]]%", '')
7352     return key .. "[[=function(m) return ]] .. ret .. [[ end]]
7353 end
7354
7355 function Babel.capt_map(from, mapno)
7356     return Babel.capture_maps[mapno][from] or from
7357 end
7358
7359 -- Handle the {n|abc|ABC} syntax in captures
7360 function Babel.capture_func_map(capno, from, to)
7361     local u = unicode.utf8
7362     from = u.gsub(from, '{{%x%x%x%x+}}',
7363         function (n)
7364             return u.char(tonumber(n, 16))
7365         end)
7366     to = u.gsub(to, '{{%x%x%x%x+}}',
7367         function (n)
7368             return u.char(tonumber(n, 16))
7369         end)
7370     local froms = {}
7371     for s in string.utfcharacters(from) do
7372         table.insert(froms, s)
7373     end
7374     local cnt = 1
7375     table.insert(Babel.capture_maps, {})
7376     local mlen = table.getn(Babel.capture_maps)
7377     for s in string.utfcharacters(to) do
7378         Babel.capture_maps[mlen][froms[cnt]] = s
7379         cnt = cnt + 1
7380     end
7381     return "]]..Babel.capt_map(m[" .. capno .. "], " ..

```

```

7382         (mlen) .. ").." .. "["
7383 end
7384
7385 -- Create/Extend reversed sorted list of kashida weights:
7386 function Babel.capture_kashida(key, wt)
7387     wt = tonumber(wt)
7388     if Babel.kashida_wts then
7389         for p, q in ipairs(Babel.kashida_wts) do
7390             if wt == q then
7391                 break
7392             elseif wt > q then
7393                 table.insert(Babel.kashida_wts, p, wt)
7394                 break
7395             elseif table.getn(Babel.kashida_wts) == p then
7396                 table.insert(Babel.kashida_wts, wt)
7397             end
7398         end
7399     else
7400         Babel.kashida_wts = { wt }
7401     end
7402     return 'kashida = ' .. wt
7403 end
7404
7405 -- Experimental: applies prehyphenation transforms to a string (letters
7406 -- and spaces).
7407 function Babel.string_prehyphenation(str, locale)
7408     local n, head, last, res
7409     head = node.new(8, 0) -- dummy (hack just to start)
7410     last = head
7411     for s in string.utfvalues(str) do
7412         if s == 20 then
7413             n = node.new(12, 0)
7414         else
7415             n = node.new(29, 0)
7416             n.char = s
7417         end
7418         node.set_attribute(n, Babel.attr_locale, locale)
7419         last.next = n
7420         last = n
7421     end
7422     head = Babel.hyphenate_replace(head, 0)
7423     res = ''
7424     for n in node.traverse(head) do
7425         if n.id == 12 then
7426             res = res .. ' '
7427         elseif n.id == 29 then
7428             res = res .. unicode.utf8.char(n.char)
7429         end
7430     end
7431     tex.print(res)
7432 end
7433 </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.

```
7434 (*basic-r)
7435 Babel = Babel or {}
7436
7437 Babel.bidi_enabled = true
7438
7439 require('babel-data-bidi.lua')
7440
7441 local characters = Babel.characters
7442 local ranges = Babel.ranges
7443
7444 local DIR = node.id("dir")
7445
7446 local function dir_mark(head, from, to, outer)
7447   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7448   local d = node.new(DIR)
7449   d.dir = '+' .. dir
7450   node.insert_before(head, from, d)
7451   d = node.new(DIR)
7452   d.dir = '-' .. dir
7453   node.insert_after(head, to, d)
7454 end
7455
7456 function Babel.bidi(head, ispar)
7457   local first_n, last_n          -- first and last char with nums
7458   local last_es                  -- an auxiliary 'last' used with nums
7459   local first_d, last_d          -- first and last char in L/R block
7460   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):

```
7461 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7462 local strong_lr = (strong == 'l') and 'l' or 'r'
7463 local outer = strong
```

```

7464
7465 local new_dir = false
7466 local first_dir = false
7467 local inmath = false
7468
7469 local last_lr
7470
7471 local type_n = ''
7472
7473 for item in node.traverse(head) do
7474
7475   -- three cases: glyph, dir, otherwise
7476   if item.id == node.id'glyph'
7477     or (item.id == 7 and item.subtype == 2) then
7478
7479     local itemchar
7480     if item.id == 7 and item.subtype == 2 then
7481       itemchar = item.replace.char
7482     else
7483       itemchar = item.char
7484     end
7485     local chardata = characters[itemchar]
7486     dir = chardata and chardata.d or nil
7487     if not dir then
7488       for nn, et in ipairs(ranges) do
7489         if itemchar < et[1] then
7490           break
7491         elseif itemchar <= et[2] then
7492           dir = et[3]
7493           break
7494         end
7495       end
7496     end
7497     dir = dir or 'l'
7498     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).

```

7499   if new_dir then
7500     attr_dir = 0
7501     for at in node.traverse(item.attr) do
7502       if at.number == Babel.attr_dir then
7503         attr_dir = at.value & 0x3
7504       end
7505     end
7506     if attr_dir == 1 then
7507       strong = 'r'
7508     elseif attr_dir == 2 then
7509       strong = 'al'
7510     else
7511       strong = 'l'
7512     end
7513     strong_lr = (strong == 'l') and 'l' or 'r'
7514     outer = strong_lr
7515     new_dir = false
7516   end
7517
7518   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7519   dir_real = dir -- We need dir_real to set strong below

```

```
7520     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:

```
7521     if strong == 'al' then
7522         if dir == 'en' then dir = 'an' end -- W2
7523         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7524         strong_lr = 'r' -- W3
7525     end
```

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

```
7526     elseif item.id == node.id'dir' and not inmath then
7527         new_dir = true
7528         dir = nil
7529     elseif item.id == node.id'math' then
7530         inmath = (item.subtype == 0)
7531     else
7532         dir = nil -- Not a char
7533     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>.

```
7534     if dir == 'en' or dir == 'an' or dir == 'et' then
7535         if dir ~= 'et' then
7536             type_n = dir
7537         end
7538         first_n = first_n or item
7539         last_n = last_es or item
7540         last_es = nil
7541     elseif dir == 'es' and last_n then -- W3+W6
7542         last_es = item
7543     elseif dir == 'cs' then -- it's right - do nothing
7544     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7545         if strong_lr == 'r' and type_n ~= '' then
7546             dir_mark(head, first_n, last_n, 'r')
7547         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7548             dir_mark(head, first_n, last_n, 'r')
7549             dir_mark(head, first_d, last_d, outer)
7550             first_d, last_d = nil, nil
7551         elseif strong_lr == 'l' and type_n ~= '' then
7552             last_d = last_n
7553         end
7554         type_n = ''
7555         first_n, last_n = nil, nil
7556     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:

```
7557     if dir == 'l' or dir == 'r' then
7558         if dir ~= outer then
7559             first_d = first_d or item
7560             last_d = item
7561         elseif first_d and dir ~= strong_lr then
7562             dir_mark(head, first_d, last_d, outer)
7563             first_d, last_d = nil, nil
7564         end
7565     end
```

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

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

```

7566     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7567         item.char = characters[item.char] and
7568             characters[item.char].m or item.char
7569     elseif (dir or new_dir) and last_lr ~= item then
7570         local mir = outer .. strong_lr .. (dir or outer)
7571         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7572             for ch in node.traverse(node.next(last_lr)) do
7573                 if ch == item then break end
7574                 if ch.id == node.id'glyph' and characters[ch.char] then
7575                     ch.char = characters[ch.char].m or ch.char
7576                 end
7577             end
7578         end
7579     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).

```

7580     if dir == 'l' or dir == 'r' then
7581         last_lr = item
7582         strong = dir_real          -- Don't search back - best save now
7583         strong_lr = (strong == 'l') and 'l' or 'r'
7584     elseif new_dir then
7585         last_lr = nil
7586     end
7587 end

```

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

```

7588 if last_lr and outer == 'r' then
7589     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7590         if characters[ch.char] then
7591             ch.char = characters[ch.char].m or ch.char
7592         end
7593     end
7594 end
7595 if first_n then
7596     dir_mark(head, first_n, last_n, outer)
7597 end
7598 if first_d then
7599     dir_mark(head, first_d, last_d, outer)
7600 end

```

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

```

7601 return node.prev(head) or head
7602 end
7603 </basic-r>

```

And here the Lua code for bidi=basic:

```

7604 <(*basic)
7605 Babel = Babel or {}
7606
7607 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7608
7609 Babel.fontmap = Babel.fontmap or {}
7610 Babel.fontmap[0] = {}      -- l
7611 Babel.fontmap[1] = {}      -- r
7612 Babel.fontmap[2] = {}      -- al/an
7613

```



```

7614 -- To cancel mirroring. Also OML, OMS, U?
7615 Babel.symbol_fonts = Babel.symbol_fonts or {}
7616 Babel.symbol_fonts[font.id('tenln')] = true
7617 Babel.symbol_fonts[font.id('tenlnw')] = true
7618 Babel.symbol_fonts[font.id('tencirc')] = true
7619 Babel.symbol_fonts[font.id('tencircw')] = true
7620
7621 Babel.bidi_enabled = true
7622 Babel.mirroring_enabled = true
7623
7624 require('babel-data-bidi.lua')
7625
7626 local characters = Babel.characters
7627 local ranges = Babel.ranges
7628
7629 local DIR = node.id('dir')
7630 local GLYPH = node.id('glyph')
7631
7632 local function insert_implicit(head, state, outer)
7633   local new_state = state
7634   if state.sim and state.eim and state.sim ~= state.eim then
7635     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7636     local d = node.new(DIR)
7637     d.dir = '+' .. dir
7638     node.insert_before(head, state.sim, d)
7639     local d = node.new(DIR)
7640     d.dir = '-' .. dir
7641     node.insert_after(head, state.eim, d)
7642   end
7643   new_state.sim, new_state.eim = nil, nil
7644   return head, new_state
7645 end
7646
7647 local function insert_numeric(head, state)
7648   local new
7649   local new_state = state
7650   if state.san and state.ean and state.san ~= state.ean then
7651     local d = node.new(DIR)
7652     d.dir = '+TLT'
7653     _, new = node.insert_before(head, state.san, d)
7654     if state.san == state.sim then state.sim = new end
7655     local d = node.new(DIR)
7656     d.dir = '-TLT'
7657     _, new = node.insert_after(head, state.ean, d)
7658     if state.ean == state.eim then state.eim = new end
7659   end
7660   new_state.san, new_state.ean = nil, nil
7661   return head, new_state
7662 end
7663
7664 local function glyph_not_symbol_font(node)
7665   if node.id == GLYPH then
7666     return not Babel.symbol_fonts[node.font]
7667   else
7668     return false
7669   end
7670 end
7671
7672 -- TODO - \hbox with an explicit dir can lead to wrong results
7673 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7674 -- was s made to improve the situation, but the problem is the 3-dir
7675 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7676 -- well.

```

```

7677
7678 function Babel.bidi(head, ispar, hdir)
7679   local d    -- d is used mainly for computations in a loop
7680   local prev_d = ''
7681   local new_d = false
7682
7683   local nodes = {}
7684   local outer_first = nil
7685   local inmath = false
7686
7687   local glue_d = nil
7688   local glue_i = nil
7689
7690   local has_en = false
7691   local first_et = nil
7692
7693   local has_hyperlink = false
7694
7695   local ATDIR = Babel.attr_dir
7696
7697   local save_outer
7698   local temp = node.get_attribute(head, ATDIR)
7699   if temp then
7700     temp = temp & 0x3
7701     save_outer = (temp == 0 and 'l') or
7702                  (temp == 1 and 'r') or
7703                  (temp == 2 and 'al')
7704   elseif ispar then      -- Or error? Shouldn't happen
7705     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7706   else                  -- Or error? Shouldn't happen
7707     save_outer = ('TRT' == hdir) and 'r' or 'l'
7708   end
7709   -- when the callback is called, we are just _after_ the box,
7710   -- and the textdir is that of the surrounding text
7711   -- if not ispar and hdir ~= tex.textdir then
7712   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7713   -- end
7714   local outer = save_outer
7715   local last = outer
7716   -- 'al' is only taken into account in the first, current loop
7717   if save_outer == 'al' then save_outer = 'r' end
7718
7719   local fontmap = Babel.fontmap
7720
7721   for item in node.traverse(head) do
7722
7723     -- In what follows, #node is the last (previous) node, because the
7724     -- current one is not added until we start processing the neutrals.
7725
7726     -- three cases: glyph, dir, otherwise
7727     if glyph_not_symbol_font(item)
7728       or (item.id == 7 and item.subtype == 2) then
7729
7730       local d_font = nil
7731       local item_r
7732       if item.id == 7 and item.subtype == 2 then
7733         item_r = item.replace    -- automatic discs have just 1 glyph
7734       else
7735         item_r = item
7736       end
7737       local chardata = characters[item_r.char]
7738       d = chardata and chardata.d or nil
7739       if not d or d == 'nsm' then

```

```

7740     for nn, et in ipairs(ranges) do
7741         if item_r.char < et[1] then
7742             break
7743         elseif item_r.char <= et[2] then
7744             if not d then d = et[3]
7745             elseif d == 'nsm' then d_font = et[3]
7746             end
7747             break
7748         end
7749     end
7750 end
7751 d = d or 'l'
7752
7753 -- A short 'pause' in bidi for mapfont
7754 d_font = d_font or d
7755 d_font = (d_font == 'l' and 0) or
7756           (d_font == 'nsm' and 0) or
7757           (d_font == 'r' and 1) or
7758           (d_font == 'al' and 2) or
7759           (d_font == 'an' and 2) or nil
7760 if d_font and fontmap and fontmap[d_font][item_r.font] then
7761     item_r.font = fontmap[d_font][item_r.font]
7762 end
7763
7764 if new_d then
7765     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7766     if inmath then
7767         attr_d = 0
7768     else
7769         attr_d = node.get_attribute(item, ATDIR)
7770         attr_d = attr_d & 0x3
7771     end
7772     if attr_d == 1 then
7773         outer_first = 'r'
7774         last = 'r'
7775     elseif attr_d == 2 then
7776         outer_first = 'r'
7777         last = 'al'
7778     else
7779         outer_first = 'l'
7780         last = 'l'
7781     end
7782     outer = last
7783     has_en = false
7784     first_et = nil
7785     new_d = false
7786 end
7787
7788 if glue_d then
7789     if (d == 'l' and 'l' or 'r') ~= glue_d then
7790         table.insert(nodes, {glue_i, 'on', nil})
7791     end
7792     glue_d = nil
7793     glue_i = nil
7794 end
7795
7796 elseif item.id == DIR then
7797     d = nil
7798
7799     if head ~= item then new_d = true end
7800
7801 elseif item.id == node.id'glue' and item.subtype == 13 then
7802     glue_d = d

```

```

7803     glue_i = item
7804     d = nil
7805
7806     elseif item.id == node.id'math' then
7807         inmath = (item.subtype == 0)
7808
7809     elseif item.id == 8 and item.subtype == 19 then
7810         has_hyperlink = true
7811
7812     else
7813         d = nil
7814     end
7815
7816     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7817     if last == 'al' and d == 'en' then
7818         d = 'an'          -- W3
7819     elseif last == 'al' and (d == 'et' or d == 'es') then
7820         d = 'on'          -- W6
7821     end
7822
7823     -- EN + CS/ES + EN      -- W4
7824     if d == 'en' and #nodes >= 2 then
7825         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7826             and nodes[#nodes-1][2] == 'en' then
7827             nodes[#nodes][2] = 'en'
7828         end
7829     end
7830
7831     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7832     if d == 'an' and #nodes >= 2 then
7833         if (nodes[#nodes][2] == 'cs')
7834             and nodes[#nodes-1][2] == 'an' then
7835             nodes[#nodes][2] = 'an'
7836         end
7837     end
7838
7839     -- ET/EN                  -- W5 + W7->l / W6->on
7840     if d == 'et' then
7841         first_et = first_et or (#nodes + 1)
7842     elseif d == 'en' then
7843         has_en = true
7844         first_et = first_et or (#nodes + 1)
7845     elseif first_et then      -- d may be nil here !
7846         if has_en then
7847             if last == 'l' then
7848                 temp = 'l'    -- W7
7849             else
7850                 temp = 'en'   -- W5
7851             end
7852         else
7853             temp = 'on'       -- W6
7854         end
7855         for e = first_et, #nodes do
7856             if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7857         end
7858         first_et = nil
7859         has_en = false
7860     end
7861
7862     -- Force mathdir in math if ON (currently works as expected only
7863     -- with 'l')
7864     if inmath and d == 'on' then
7865         d = ('TRT' == tex.mathdir) and 'r' or 'l'

```

```

7866     end
7867
7868     if d then
7869         if d == 'al' then
7870             d = 'r'
7871             last = 'al'
7872         elseif d == 'l' or d == 'r' then
7873             last = d
7874         end
7875         prev_d = d
7876         table.insert(nodes, {item, d, outer_first})
7877     end
7878
7879     outer_first = nil
7880
7881 end
7882
7883 -- TODO -- repeated here in case EN/ET is the last node. Find a
7884 -- better way of doing things:
7885 if first_et then      -- dir may be nil here !
7886     if has_en then
7887         if last == 'l' then
7888             temp = 'l'      -- W7
7889         else
7890             temp = 'en'     -- W5
7891         end
7892     else
7893         temp = 'on'        -- W6
7894     end
7895     for e = first_et, #nodes do
7896         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7897     end
7898 end
7899
7900 -- dummy node, to close things
7901 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7902
7903 ----- NEUTRAL -----
7904
7905 outer = save_outer
7906 last = outer
7907
7908 local first_on = nil
7909
7910 for q = 1, #nodes do
7911     local item
7912
7913     local outer_first = nodes[q][3]
7914     outer = outer_first or outer
7915     last = outer_first or last
7916
7917     local d = nodes[q][2]
7918     if d == 'an' or d == 'en' then d = 'r' end
7919     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7920
7921     if d == 'on' then
7922         first_on = first_on or q
7923     elseif first_on then
7924         if last == d then
7925             temp = d
7926         else
7927             temp = outer
7928         end

```

```

7929     for r = first_on, q - 1 do
7930         nodes[r][2] = temp
7931         item = nodes[r][1]    -- MIRRORING
7932         if Babel.mirroring_enabled and glyph_not_symbol_font(item)
7933             and temp == 'r' and characters[item.char] then
7934             local font_mode = ''
7935             if item.font > 0 and font.fonts[item.font].properties then
7936                 font_mode = font.fonts[item.font].properties.mode
7937             end
7938             if font_mode ~= 'harf' and font_mode ~= 'plug' then
7939                 item.char = characters[item.char].m or item.char
7940             end
7941         end
7942     end
7943     first_on = nil
7944 end
7945
7946 if d == 'r' or d == 'l' then last = d end
7947 end
7948
7949 ----- IMPLICIT, REORDER -----
7950
7951 outer = save_outer
7952 last = outer
7953
7954 local state = {}
7955 state.has_r = false
7956
7957 for q = 1, #nodes do
7958
7959     local item = nodes[q][1]
7960
7961     outer = nodes[q][3] or outer
7962
7963     local d = nodes[q][2]
7964
7965     if d == 'nsm' then d = last end          -- W1
7966     if d == 'en' then d = 'an' end
7967     local isdir = (d == 'r' or d == 'l')
7968
7969     if outer == 'l' and d == 'an' then
7970         state.san = state.san or item
7971         state.ean = item
7972     elseif state.san then
7973         head, state = insert_numeric(head, state)
7974     end
7975
7976     if outer == 'l' then
7977         if d == 'an' or d == 'r' then      -- im -> implicit
7978             if d == 'r' then state.has_r = true end
7979             state.sim = state.sim or item
7980             state.eim = item
7981         elseif d == 'l' and state.sim and state.has_r then
7982             head, state = insert_implicit(head, state, outer)
7983         elseif d == 'l' then
7984             state.sim, state.eim, state.has_r = nil, nil, false
7985         end
7986     else
7987         if d == 'an' or d == 'l' then
7988             if nodes[q][3] then -- nil except after an explicit dir
7989                 state.sim = item -- so we move sim 'inside' the group
7990             else
7991                 state.sim = state.sim or item

```

```

7992         end
7993         state.eim = item
7994     elseif d == 'r' and state.sim then
7995         head, state = insert_implicit(head, state, outer)
7996     elseif d == 'r' then
7997         state.sim, state.eim = nil, nil
7998     end
7999 end
8000
8001 if isdir then
8002     last = d          -- Don't search back - best save now
8003 elseif d == 'on' and state.san then
8004     state.san = state.san or item
8005     state.ean = item
8006 end
8007
8008 end
8009
8010 head = node.prev(head) or head
8011
8012 ----- FIX HYPERLINKS -----
8013
8014 if has_hyperlink then
8015     local flag, linking = 0, 0
8016     for item in node.traverse(head) do
8017         if item.id == DIR then
8018             if item.dir == '+TRT' or item.dir == '+TLT' then
8019                 flag = flag + 1
8020             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8021                 flag = flag - 1
8022             end
8023         elseif item.id == 8 and item.subtype == 19 then
8024             linking = flag
8025         elseif item.id == 8 and item.subtype == 20 then
8026             if linking > 0 then
8027                 if item.prev.id == DIR and
8028                     (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8029                     d = node.new(DIR)
8030                     d.dir = item.prev.dir
8031                     node.remove(head, item.prev)
8032                     node.insert_after(head, item, d)
8033                 end
8034             end
8035             linking = 0
8036         end
8037     end
8038 end
8039
8040 return head
8041 end
8042 </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.

```
8043 ⟨*nil⟩
8044 \ProvidesLanguage{nil}[\⟨date⟩ v\⟨version⟩ Nil language]
8045 \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.

```
8046 \ifx\l@nil\undefined
8047   \newlanguage\l@nil
8048   \@namedef{bbl@hyphendata@the\l@nil}{\{\}\}% Remove warning
8049   \let\bbl@elt\relax
8050   \edef\bbl@languages{% Add it to the list of languages
8051     \bbl@languages\bbl@elt{nil}{the\l@nil}\{\}\}
8052 \fi
```

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

```
8053 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
8054 \let\captionnil\empty
8055 \let\datenil\empty
```

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

```
8056 \def\bbl@inidata@nil{%
8057   \bbl@elt{identification}{tag.ini}{und}%
8058   \bbl@elt{identification}{load.level}{0}%
8059   \bbl@elt{identification}{charset}{utf8}%
8060   \bbl@elt{identification}{version}{1.0}%
8061   \bbl@elt{identification}{date}{2022-05-16}%
8062   \bbl@elt{identification}{name.local}{nil}%
8063   \bbl@elt{identification}{name.english}{nil}%
8064   \bbl@elt{identification}{name.babel}{nil}%
8065   \bbl@elt{identification}{tag.bcp47}{und}%
8066   \bbl@elt{identification}{language.tag.bcp47}{und}%
8067   \bbl@elt{identification}{tag.opentype}{dflt}%
8068   \bbl@elt{identification}{script.name}{Latin}%
8069   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8070   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8071   \bbl@elt{identification}{level}{1}%
8072   \bbl@elt{identification}{encodings}{\}%
8073   \bbl@elt{identification}{derivate}{no}}
8074 \@namedef{bbl@tbcp@nil}{und}
8075 \@namedef{bbl@lbcp@nil}{und}
8076 \@namedef{bbl@ casing@nil}{und} % TODO
8077 \@namedef{bbl@lotf@nil}{dflt}
8078 \@namedef{bbl@elname@nil}{nil}
8079 \@namedef{bbl@lname@nil}{nil}
8080 \@namedef{bbl@esname@nil}{Latin}
8081 \@namedef{bbl@sname@nil}{Latin}
8082 \@namedef{bbl@sbc@nil}{Latn}
8083 \@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.

```
8084 \ldf@finish{nil}
8085 \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.

```
8086 <<Compute Julian day>> ≡
8087 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8088 \def\bbl@cs@gregleap#1{%
8089   (\bbl@fpmmod{#1}{4} == 0) &&
8090   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8091 \def\bbl@cs@jd#1#2#3{% year, month, day
8092   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8093     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8094     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8095     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }
8096 <</Compute Julian day>>
```

### 13.1 Islamic

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

```
8097 <ca-islamic>
8098 \ExplSyntaxOn
8099 <<Compute Julian day>>
8100 % == islamic (default)
8101 % Not yet implemented
8102 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar:

```
8103 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8104   ((#3 + ceil(29.5 * (#2 - 1)) +
8105     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8106     1948439.5) - 1) }
8107 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8108 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8109 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8110 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8111 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8112 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
8113   \edef\bbl@tempa{%
8114     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8115   \edef#5{%
8116     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8117   \edef#6{\fp_eval:n{
8118     min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8119   \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).

```
8120 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8121 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8122 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8123 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8124 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
```

```

8125 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8126 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8127 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8128 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8129 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8130 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8131 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8132 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8133 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8134 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8135 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8136 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8137 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8138 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8139 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8140 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8141 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8142 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8143 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8144 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8145 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8146 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8147 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8148 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8149 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8150 65401,65431,65460,65490,65520}
8151 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8152 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8153 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8154 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8155 \ifnum#2>2014 \ifnum#2<2038
8156 \bbl@afterfi\expandafter\@gobble
8157 \fi\fi
8158 {\bbl@error{year-out-range}{2014-2038}{}}}%
8159 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8160 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8161 \count@\@ne
8162 \bbl@foreach\bbl@cs@umalqura@data{%
8163 \advance\count@\@ne
8164 \ifnum##1>\bbl@tempd\else
8165 \edef\bbl@tempe{\the\count@}%
8166 \edef\bbl@tempb{##1}%
8167 \fi}%
8168 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8169 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8170 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8171 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8172 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8173 \ExplSyntaxOff
8174 \bbl@add\bbl@precalendar{%
8175 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8176 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
8177 \bbl@replace\bbl@ld@calendar{+}{}}%
8178 \bbl@replace\bbl@ld@calendar{-}{}}%
8179 /ca-islamic)

```

## 13.2 Hebrew

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

```

8180 (*ca-hebrew)
8181 \newcount\bbl@cntcommon

```

```

8182 \def\bbl@remainder#1#2#3{%
8183   #3=#1\relax
8184   \divide #3 by #2\relax
8185   \multiply #3 by -#2\relax
8186   \advance #3 by #1\relax}%
8187 \newif\ifbbl@divisible
8188 \def\bbl@checkifdivisible#1#2{%
8189   {\countdef\tmp=0
8190    \bbl@remainder{#1}{#2}{\tmp}%
8191    \ifnum \tmp=0
8192      \global\bbl@divisibletrue
8193    \else
8194      \global\bbl@divisiblefalse
8195    \fi}}
8196 \newif\ifbbl@gregleap
8197 \def\bbl@ifgregleap#1{%
8198   \bbl@checkifdivisible{#1}{4}%
8199   \ifbbl@divisible
8200     \bbl@checkifdivisible{#1}{100}%
8201     \ifbbl@divisible
8202       \bbl@checkifdivisible{#1}{400}%
8203       \ifbbl@divisible
8204         \bbl@gregleaptrue
8205       \else
8206         \bbl@gregleapfalse
8207       \fi
8208     \else
8209       \bbl@gregleaptrue
8210     \fi
8211   \else
8212     \bbl@gregleapfalse
8213   \fi
8214   \ifbbl@gregleap}
8215 \def\bbl@gregdayspriormonths#1#2#3{%
8216   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8217     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8218   \bbl@ifgregleap{#2}%
8219   \ifnum #1 > 2
8220     \advance #3 by 1
8221   \fi
8222   \fi
8223   \global\bbl@cntcommon=#3}%
8224   #3=\bbl@cntcommon}
8225 \def\bbl@gregdaysprioryears#1#2{%
8226   {\countdef\tmpc=4
8227    \countdef\tmpb=2
8228    \tmpb=#1\relax
8229    \advance \tmpb by -1
8230    \tmpc=\tmpb
8231    \multiply \tmpc by 365
8232    #2=\tmpc
8233    \tmpc=\tmpb
8234    \divide \tmpc by 4
8235    \advance #2 by \tmpc
8236    \tmpc=\tmpb
8237    \divide \tmpc by 100
8238    \advance #2 by -\tmpc
8239    \tmpc=\tmpb
8240    \divide \tmpc by 400
8241    \advance #2 by \tmpc
8242    \global\bbl@cntcommon=#2\relax}%
8243   #2=\bbl@cntcommon}
8244 \def\bbl@absfromgreg#1#2#3#4{%

```

```

8245 {\countdef\tmpd=0
8246 #4=#1\relax
8247 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8248 \advance #4 by \tmpd
8249 \bbl@gregdaysprioryears{#3}{\tmpd}%
8250 \advance #4 by \tmpd
8251 \global\bbl@cntcommon=#4\relax}%
8252 #4=\bbl@cntcommon}
8253 \newif\ifbbl@hebrleap
8254 \def\bbl@checkleaphebrewyear#1{%
8255 {\countdef\tmpa=0
8256 \countdef\tmpb=1
8257 \tmpa=#1\relax
8258 \multiply \tmpa by 7
8259 \advance \tmpa by 1
8260 \bbl@remainder{\tmpa}{19}{\tmpb}%
8261 \ifnum \tmpb < 7
8262 \global\bbl@hebrleaptrue
8263 \else
8264 \global\bbl@hebrleapfalse
8265 \fi}}
8266 \def\bbl@hebrelapsedmonths#1#2{%
8267 {\countdef\tmpa=0
8268 \countdef\tmpb=1
8269 \countdef\tmpc=2
8270 \tmpa=#1\relax
8271 \advance \tmpa by -1
8272 #2=\tmpa
8273 \divide #2 by 19
8274 \multiply #2 by 235
8275 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8276 \tmpc=\tmpb
8277 \multiply \tmpb by 12
8278 \advance #2 by \tmpb
8279 \multiply \tmpc by 7
8280 \advance \tmpc by 1
8281 \divide \tmpc by 19
8282 \advance #2 by \tmpc
8283 \global\bbl@cntcommon=#2}%
8284 #2=\bbl@cntcommon}
8285 \def\bbl@hebrelapseddays#1#2{%
8286 {\countdef\tmpa=0
8287 \countdef\tmpb=1
8288 \countdef\tmpc=2
8289 \bbl@hebrelapsedmonths{#1}{#2}%
8290 \tmpa=#2\relax
8291 \multiply \tmpa by 13753
8292 \advance \tmpa by 5604
8293 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8294 \divide \tmpa by 25920
8295 \multiply #2 by 29
8296 \advance #2 by 1
8297 \advance #2 by \tmpa
8298 \bbl@remainder{#2}{7}{\tmpa}%
8299 \ifnum \tmpc < 19440
8300 \ifnum \tmpc < 9924
8301 \else
8302 \ifnum \tmpa=2
8303 \bbl@checkleaphebrewyear{#1}% of a common year
8304 \ifbbl@hebrleap
8305 \else
8306 \advance #2 by 1
8307 \fi

```

```

8308         \fi
8309     \fi
8310     \ifnum \tmpc < 16789
8311     \else
8312         \ifnum \tmpa=1
8313             \advance #1 by -1
8314             \bbl@checkleaphebyear{#1}% at the end of leap year
8315             \ifbbl@hebrleap
8316                 \advance #2 by 1
8317             \fi
8318         \fi
8319     \fi
8320 \else
8321     \advance #2 by 1
8322 \fi
8323 \bbl@remainder{#2}{7}{\tmpa}%
8324 \ifnum \tmpa=0
8325     \advance #2 by 1
8326 \else
8327     \ifnum \tmpa=3
8328         \advance #2 by 1
8329     \else
8330         \ifnum \tmpa=5
8331             \advance #2 by 1
8332         \fi
8333     \fi
8334 \fi
8335 \global\bbl@cntcommon=#2\relax}%
8336 #2=\bbl@cntcommon}
8337 \def\bbl@daysinhebyear#1#2{%
8338     {\countdef\tmpe=12
8339     \bbl@hebreleapseddays{#1}{\tmpe}%
8340     \advance #1 by 1
8341     \bbl@hebreleapseddays{#1}{#2}%
8342     \advance #2 by -\tmpe
8343     \global\bbl@cntcommon=#2}%
8344 #2=\bbl@cntcommon}
8345 \def\bbl@hebrdayspriormonths#1#2#3{%
8346     {\countdef\tmpf= 14
8347     #3=\ifcase #1\relax
8348         0 \or
8349         0 \or
8350         30 \or
8351         59 \or
8352         89 \or
8353         118 \or
8354         148 \or
8355         148 \or
8356         177 \or
8357         207 \or
8358         236 \or
8359         266 \or
8360         295 \or
8361         325 \or
8362         400
8363     \fi
8364     \bbl@checkleaphebyear{#2}%
8365     \ifbbl@hebrleap
8366         \ifnum #1 > 6
8367             \advance #3 by 30
8368         \fi
8369     \fi
8370     \bbl@daysinhebyear{#2}{\tmpf}%

```

```

8371 \ifnum #1 > 3
8372 \ifnum \tmpf=353
8373 \advance #3 by -1
8374 \fi
8375 \ifnum \tmpf=383
8376 \advance #3 by -1
8377 \fi
8378 \fi
8379 \ifnum #1 > 2
8380 \ifnum \tmpf=355
8381 \advance #3 by 1
8382 \fi
8383 \ifnum \tmpf=385
8384 \advance #3 by 1
8385 \fi
8386 \fi
8387 \global\bbl@cntcommon=#3\relax}%
8388 #3=\bbl@cntcommon}
8389 \def\bbl@absfromhebr#1#2#3#4{%
8390 {#4=#1\relax
8391 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8392 \advance #4 by #1\relax
8393 \bbl@hebreleapseddays{#3}{#1}%
8394 \advance #4 by #1\relax
8395 \advance #4 by -1373429
8396 \global\bbl@cntcommon=#4\relax}%
8397 #4=\bbl@cntcommon}
8398 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8399 {\countdef\tmpx= 17
8400 \countdef\tmpy= 18
8401 \countdef\tmpz= 19
8402 #6=#3\relax
8403 \global\advance #6 by 3761
8404 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8405 \tmpz=1 \tmpy=1
8406 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8407 \ifnum \tmpx > #4\relax
8408 \global\advance #6 by -1
8409 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8410 \fi
8411 \advance #4 by -\tmpx
8412 \advance #4 by 1
8413 #5=#4\relax
8414 \divide #5 by 30
8415 \loop
8416 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8417 \ifnum \tmpx < #4\relax
8418 \advance #5 by 1
8419 \tmpy=\tmpx
8420 \repeat
8421 \global\advance #5 by -1
8422 \global\advance #4 by -\tmpy}}
8423 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8424 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8425 \def\bbl@ca@hebrew#1-#2-#3@@#4#5#6{%
8426 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8427 \bbl@hebrfromgreg
8428 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8429 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8430 \edef#4{\the\bbl@hebyear}%
8431 \edef#5{\the\bbl@hebrmonth}%
8432 \edef#6{\the\bbl@hebrday}}
8433 </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).

```
8434 (*ca-persian)
8435 \ExplSyntaxOn
8436 <<Compute Julian day>>
8437 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8438   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8439 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8440   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8441   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8442     \bbl@afterfi\expandafter@gobble
8443   \fi\fi
8444   {\bbl@error{year-out-range}{2013-2050}{}}}%
8445   \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8446   \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8447   \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8448   \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8449   \ifnum\bbl@tempc<\bbl@tempb
8450     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8451     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8452     \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8453     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8454   \fi
8455   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8456   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8457   \edef#5{\fp_eval:n{% set Jalali month
8458     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8459   \edef#6{\fp_eval:n{% set Jalali day
8460     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8461 \ExplSyntaxOff
8462 </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.

```
8463 (*ca-coptic)
8464 \ExplSyntaxOn
8465 <<Compute Julian day>>
8466 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8467   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8468   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8469   \edef#4{\fp_eval:n{%
8470     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8471   \edef\bbl@tempc{\fp_eval:n{%
8472     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8473   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8474   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8475 \ExplSyntaxOff
8476 </ca-coptic>
8477 (*ca-ethiopic)
8478 \ExplSyntaxOn
8479 <<Compute Julian day>>
8480 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8481   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8482   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8483   \edef#4{\fp_eval:n{%
8484     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
```

```

8485 \edef\bbl@tempc{\fp_eval:n{%
8486 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8487 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8488 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8489 \ExplSyntaxOff
8490 \</ca-ethiopic>

```

### 13.5 Buddhist

That's very simple.

```

8491 \<ca-buddhist>
8492 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8493 \edef#4{\number\numexpr#1+543\relax}%
8494 \edef#5{#2}%
8495 \edef#6{#3}}
8496 \</ca-buddhist>
8497 %
8498 % \subsection{Chinese}
8499 %
8500 % Brute force, with the Julian day of first day of each month. The
8501 % table has been computed with the help of \textsf{python-lunardate} by
8502 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8503 % is 2015-2044.
8504 %
8505 % \begin{macrocode}
8506 \<ca-chinese>
8507 \ExplSyntaxOn
8508 \<<Compute Julian day>>
8509 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8510 \edef\bbl@tempd{\fp_eval:n{%
8511 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8512 \count@ \z@
8513 \@tempcnta=2015
8514 \bbl@foreach\bbl@cs@chinese@data{%
8515 \ifnum##1>\bbl@tempd\else
8516 \advance\count@\@ne
8517 \ifnum\count@>12
8518 \count@\@ne
8519 \advance\@tempcnta\@ne\fi
8520 \bbl@xin@{,##1,}{, \bbl@cs@chinese@leap,}%
8521 \ifin@
8522 \advance\count@\m@ne
8523 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8524 \else
8525 \edef\bbl@tempe{\the\count@}%
8526 \fi
8527 \edef\bbl@tempb{##1}%
8528 \fi}%
8529 \edef#4{\the\@tempcnta}%
8530 \edef#5{\bbl@tempe}%
8531 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8532 \def\bbl@cs@chinese@leap{%
8533 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8534 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8535 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8536 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8537 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8538 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8539 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8540 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8541 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8542 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8543 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%

```



```

8544 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8545 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8546 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8547 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8548 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8549 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8550 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8551 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8552 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8553 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8554 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8555 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8556 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8557 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8558 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8559 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8560 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8561 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8562 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8563 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8564 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8565 10896,10926,10956,10986,11015,11045,11074,11103}
8566 \ExplSyntaxOff
8567 </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`.

```

8568 (*bplain | blplain)
8569 \catcode`\{=1 % left brace is begin-group character
8570 \catcode`\}=2 % right brace is end-group character
8571 \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).

```

8572 \openin 0 hyphen.cfg
8573 \ifeof0
8574 \else
8575   \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.

```

8576 \def\input #1 {%
8577   \let\input\input
8578   \a hyphen.cfg

```

```

8579 \let\undefined
8580 }
8581 \fi
8582 </bplain | bplain>

```

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

```

8583 <bplain>\a plain.tex
8584 <bplain>\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.

```

8585 <bplain>\def\fmtname{babel-plain}
8586 <bplain>\def\fmtname{babel-lplain}

```

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

## 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).

```

8587 <<*Emulate LaTeX>> ≡
8588 \def\@empty{}
8589 \def\loadlocalcfg#1{%
8590   \openin0#1.cfg
8591   \ifeof0
8592     \closein0
8593   \else
8594     \closein0
8595     {\immediate\writel6{*****}%
8596      \immediate\writel6{* Local config file #1.cfg used}%
8597      \immediate\writel6{*}%
8598     }
8599     \input #1.cfg\relax
8600 \fi
8601 \@endofldf}

```

## 14.3 General tools

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

```

8602 \long\def\@firstofone#1{#1}
8603 \long\def\@firstoftwo#1#2{#1}
8604 \long\def\@secondoftwo#1#2{#2}
8605 \def\@nnil{\@nil}
8606 \def\@gobbletwo#1#2{}
8607 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8608 \def\@star@or@long#1{%
8609   \@ifstar
8610   {\let\l@ngrel@x\relax#1}%
8611   {\let\l@ngrel@x\long#1}}
8612 \let\l@ngrel@x\relax
8613 \def\@car#1#2\@nil{#1}
8614 \def\@cdr#1#2\@nil{#2}
8615 \let\@typeset@protect\relax
8616 \let\protected@edef\edef
8617 \long\def\@gobble#1{}
8618 \edef\@backslashchar{\expandafter\@gobble\string\}
8619 \def\strip@prefix#1>{}
8620 \def\g@addto@macro#1#2{%

```

```

8621 \toks@{\expandafter{#1#2}%
8622 \xdef#1{\the\toks@}}
8623 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8624 \def\@nameuse#1{\csname #1\endcsname}
8625 \def\@ifundefined#1{%
8626 \expandafter\ifx\csname#1\endcsname\relax
8627 \expandafter\@firstoftwo
8628 \else
8629 \expandafter\@secondoftwo
8630 \fi}
8631 \def\@expandtwoargs#1#2#3{%
8632 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8633 \def\zap@space#1 #2{%
8634 #1%
8635 \ifx#2\@empty\else\expandafter\zap@space\fi
8636 #2}
8637 \let\bbl@trace\@gobble
8638 \def\bbl@error#1{% Implicit #2#3#4
8639 \begingroup
8640 \catcode`\:=0 \catcode`\==12 \catcode`\`=12
8641 \catcode`\^M=5 \catcode`\%=14
8642 \input errbabel.def
8643 \endgroup
8644 \bbl@error{#1}}
8645 \def\bbl@warning#1{%
8646 \begingroup
8647 \newlinechar=`^^J
8648 \def\{^^J(babel) }%
8649 \message{\{#1}%
8650 \endgroup}
8651 \let\bbl@infowarn\bbl@warning
8652 \def\bbl@info#1{%
8653 \begingroup
8654 \newlinechar=`^^J
8655 \def\{^^J}%
8656 \wlog{#1}%
8657 \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}`.

```

8658 \ifx\@preamblecmds\undefined
8659 \def\@preamblecmds{}
8660 \fi
8661 \def\@onlypreamble#1{%
8662 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8663 \@preamblecmds\do#1}}
8664 \@onlypreamble\@onlypreamble

```

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

```

8665 \def\begindocument{%
8666 \@begindocumenthook
8667 \global\let\@begindocumenthook\undefined
8668 \def\do##1{\global\let##1\undefined}%
8669 \@preamblecmds
8670 \global\let\do\noexpand}

8671 \ifx\@begindocumenthook\undefined
8672 \def\@begindocumenthook{}
8673 \fi
8674 \@onlypreamble\@begindocumenthook
8675 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8676 \def\AtEndOfPackage#1{\g@addto@macro\@endofldef{#1}}
8677 \@onlypreamble\AtEndOfPackage
8678 \def\@endofldef{}
8679 \@onlypreamble\@endofldef
8680 \let\bbl@afterlang\@empty
8681 \chardef\bbl@opt@hyphenmap\z@

```

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

```

8682 \catcode`\&=\z@
8683 \ifx&\if@files\@undefined
8684   \expandafter\let\csname if@files\expandafter\endcsname
8685     \csname iffalse\endcsname
8686 \fi
8687 \catcode`\&=4

```

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

```

8688 \def\newcommand{\@star@or@long\new@command}
8689 \def\new@command#1{%
8690   \@testopt{\@newcommand#1}0}
8691 \def\@newcommand#1[#2]{%
8692   \@ifnextchar [{\@xargdef#1[#2]}%
8693     {\@argdef#1[#2]}}
8694 \long\def\@argdef#1[#2]#3{%
8695   \@yargdef#1\@ne{#2}{#3}}
8696 \long\def\@xargdef#1[#2][#3]#4{%
8697   \expandafter\def\expandafter#1\expandafter{%
8698     \expandafter\@protected@testopt\expandafter #1%
8699     \csname\string#1\expandafter\endcsname{#3}}%
8700   \expandafter\@yargdef \csname\string#1\endcsname
8701   \tw@{#2}{#4}}
8702 \long\def\@yargdef#1#2#3{%
8703   \@tempcnta#3\relax
8704   \advance \@tempcnta \@ne
8705   \let\@hash@\relax
8706   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8707   \@tempcntb #2%
8708   \@whilenum\@tempcntb <\@tempcnta
8709   \do{%
8710     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
8711     \advance\@tempcntb \@ne}%
8712   \let\@hash@##%
8713   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8714 \def\providecommand{\@star@or@long\provide@command}
8715 \def\provide@command#1{%
8716   \begingroup
8717     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8718   \endgroup
8719   \expandafter\@ifundefined\@gtempa
8720     {\def\reserved@a{\new@command#1}}%
8721     {\let\reserved@a\relax
8722     \def\reserved@a{\new@command\reserved@a}}%
8723   \reserved@a}%
8724 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8725 \def\declare@robustcommand#1{%
8726   \edef\reserved@a{\string#1}%
8727   \def\reserved@b{#1}%
8728   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8729   \edef#1{%
8730     \ifx\reserved@a\reserved@b
8731       \noexpand\x@protect
8732     \noexpand#1%

```

```

8733 \fi
8734 \noexpand\protect
8735 \expandafter\noexpand\csname
8736 \expandafter\@gobble\string#1 \endcsname
8737 }%
8738 \expandafter\new@command\csname
8739 \expandafter\@gobble\string#1 \endcsname
8740 }
8741 \def\x@protect#1{%
8742 \ifx\protect\@typeset@protect\else
8743 \x@protect#1%
8744 \fi
8745 }
8746 \catcode`\&=\z@ % Trick to hide conditionals
8747 \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`.

```

8748 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8749 \catcode`\&=4
8750 \ifx\in@\@undefined
8751 \def\in@#1#2{%
8752 \def\in@##1#1##2##3\in@{%
8753 \ifx\in@##2\in@false\else\in@true\fi}%
8754 \in@#2#1\in@\in@}
8755 \else
8756 \let\bbl@tempa\@empty
8757 \fi
8758 \bbl@tempa

```

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

```

8759 \def\@ifpackagewith#1#2#3#4{#3}

```

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

```

8760 \def\@ifl@aded#1#2#3#4{}

```

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

```

8761 \ifx\@tempcnta\@undefined
8762 \csname newcount\endcsname\@tempcnta\relax
8763 \fi
8764 \ifx\@tempcntb\@undefined
8765 \csname newcount\endcsname\@tempcntb\relax
8766 \fi

```

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

```

8767 \ifx\bye\@undefined
8768 \advance\count10 by -2\relax
8769 \fi
8770 \ifx\@ifnextchar\@undefined
8771 \def\@ifnextchar#1#2#3{%
8772 \let\reserved@d=#1%
8773 \def\reserved@a{#2}\def\reserved@b{#3}%
8774 \futurelet\@let@token\@ifnch}
8775 \def\@ifnch{%
8776 \ifx\@let@token\@sptoken

```

```

8777     \let\reserved@c\@xifnch
8778   \else
8779     \ifx\@let@token\reserved@
8780       \let\reserved@c\reserved@a
8781     \else
8782       \let\reserved@c\reserved@b
8783     \fi
8784   \fi
8785   \reserved@c}
8786 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
8787 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8788 \fi
8789 \def\@testopt#1#2{%
8790   \@ifnextchar[#{#1}{#1[#2]}}
8791 \def\@protected@testopt#1{%
8792   \ifx\protect\@typeset@protect
8793     \expandafter\@testopt
8794   \else
8795     \@x@protect#1%
8796   \fi}
8797 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8798   #2\relax}\fi}
8799 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8800   \else\expandafter\@gobble\fi{#1}}

```

## 14.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  environment.

```

8801 \def\DeclareTextCommand{%
8802   \@dec@text@cmd\providecommand
8803 }
8804 \def\ProvideTextCommand{%
8805   \@dec@text@cmd\providecommand
8806 }
8807 \def\DeclareTextSymbol#1#2#3{%
8808   \@dec@text@cmd\chardef#1{#2}#3\relax
8809 }
8810 \def\@dec@text@cmd#1#2#3{%
8811   \expandafter\def\expandafter#2%
8812     \expandafter{%
8813       \csname#3-cmd\expandafter\endcsname
8814       \expandafter#2%
8815       \csname#3\string#2\endcsname
8816     }%
8817 %   \let\@ifdefinable\@rc@ifdefinable
8818   \expandafter#1\csname#3\string#2\endcsname
8819 }
8820 \def\@current@cmd#1{%
8821   \ifx\protect\@typeset@protect\else
8822     \noexpand#1\expandafter\@gobble
8823   \fi
8824 }
8825 \def\@changed@cmd#1#2{%
8826   \ifx\protect\@typeset@protect
8827     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8828       \expandafter\ifx\csname ?\string#1\endcsname\relax
8829         \expandafter\def\csname ?\string#1\endcsname{%
8830           \@changed@x@err{#1}%
8831         }%
8832       \fi
8833     \global\expandafter\let
8834       \csname\cf@encoding \string#1\expandafter\endcsname
8835       \csname ?\string#1\endcsname

```

```

8836     \fi
8837     \csname\cf@encoding\string#1%
8838     \expandafter\endcsname
8839 \else
8840     \noexpand#1%
8841 \fi
8842 }
8843 \def\@changed@x@err#1{%
8844     \errhelp{Your command will be ignored, type <return> to proceed}%
8845     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8846 \def\DeclareTextCommandDefault#1{%
8847     \DeclareTextCommand#1?%
8848 }
8849 \def\ProvideTextCommandDefault#1{%
8850     \ProvideTextCommand#1?%
8851 }
8852 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8853 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8854 \def\DeclareTextAccent#1#2#3{%
8855     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8856 }
8857 \def\DeclareTextCompositeCommand#1#2#3#4{%
8858     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8859     \edef\reserved@b{\string##1}%
8860     \edef\reserved@c{%
8861         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8862     \ifx\reserved@b\reserved@c
8863         \expandafter\expandafter\expandafter\ifx
8864             \expandafter\@car\reserved@a\relax\relax\@nil
8865             \@text@composite
8866         \else
8867             \edef\reserved@b##1{%
8868                 \def\expandafter\noexpand
8869                     \csname#2\string#1\endcsname###1{%
8870                     \noexpand\@text@composite
8871                         \expandafter\noexpand\csname#2\string#1\endcsname
8872                         ###1\noexpand\@empty\noexpand\@text@composite
8873                         {##1}%
8874                     }%
8875                 }%
8876             \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8877         \fi
8878         \expandafter\def\csname\expandafter\string\csname
8879             #2\endcsname\string#1-\string#3\endcsname{#4}
8880     \else
8881         \errhelp{Your command will be ignored, type <return> to proceed}%
8882         \errmessage{\string\DeclareTextCompositeCommand\space used on
8883             inappropriate command \protect#1}
8884     \fi
8885 }
8886 \def\@text@composite#1#2#3\@text@composite{%
8887     \expandafter\@text@composite@x
8888     \csname\string#1-\string#2\endcsname
8889 }
8890 \def\@text@composite@x#1#2{%
8891     \ifx#1\relax
8892         #2%
8893     \else
8894         #1%
8895     \fi
8896 }
8897 %
8898 \def\@strip@args#1:#2-#3\@strip@args{#2}

```

```

8899 \def\DeclareTextComposite#1#2#3#4{%
8900   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8901   \bgroup
8902     \lccode`\@=#4%
8903     \lowercase{%
8904       \egroup
8905       \reserved@a @%
8906     }%
8907 }
8908 %
8909 \def\UseTextSymbol#1#2{#2}
8910 \def\UseTextAccent#1#2#3{}
8911 \def\@use@text@encoding#1{}
8912 \def\DeclareTextSymbolDefault#1#2{%
8913   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8914 }
8915 \def\DeclareTextAccentDefault#1#2{%
8916   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8917 }
8918 \def\cf@encoding{OT1}

```

Currently we only use the  $\TeX$  2<sub>ε</sub> method for accents for those that are known to be made active in *some* language definition file.

```

8919 \DeclareTextAccent{"}{OT1}{127}
8920 \DeclareTextAccent{'}{OT1}{19}
8921 \DeclareTextAccent{^}{OT1}{94}
8922 \DeclareTextAccent{`}{OT1}{18}
8923 \DeclareTextAccent{~}{OT1}{126}

```

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

```

8924 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8925 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`"}
8926 \DeclareTextSymbol{\textquoteleft}{OT1}{``'}
8927 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8928 \DeclareTextSymbol{\i}{OT1}{16}
8929 \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`.

```

8930 \ifx\scriptsize\@undefined
8931   \let\scriptsize\sevenrm
8932 \fi

```

And a few more “dummy” definitions.

```

8933 \def\language{english}%
8934 \let\bbl@opt@shorthands\@nnil
8935 \def\bbl@ifshorthand#1#2#3{#2}%
8936 \let\bbl@language@opts\@empty
8937 \let\bbl@ensureinfo\@gobble
8938 \let\bbl@provide@locale\relax
8939 \ifx\babeloptionstrings\@undefined
8940   \let\bbl@opt@strings\@nnil
8941 \else
8942   \let\bbl@opt@strings\babeloptionstrings
8943 \fi
8944 \def\BabelStringsDefault{generic}
8945 \def\bbl@tempa{normal}
8946 \ifx\babeloptionmath\bbl@tempa
8947   \def\bbl@mathnormal{\noexpand\textormath}
8948 \fi
8949 \def\AfterBabelLanguage#1#2{}
8950 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8951 \let\bbl@afterlang\relax
8952 \def\bbl@opt@safe{BR}

```



```

8953 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8954 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8955 \expandafter\newif\csname ifbbl@single\endcsname
8956 \chardef\bbl@bidimode\z@
8957 <</Emulate LaTeX>>

```

A proxy file:

```

8958 <*plain>
8959 \input babel.def
8960 </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  $\TeX$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\TeX$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German  $\TeX$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International  $\TeX$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\TeX$* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).