

# Babel

## Code

Version 25.8.86174  
2025/05/10

Javier Bezos

Current maintainer

Johannes L. Braams

Original author

Localization and  
internationalization

Unicode

T<sub>E</sub>X

LuaT<sub>E</sub>X

pdfT<sub>E</sub>X

XeT<sub>E</sub>X

# Contents

<b>1</b>	<b>Identification and loading of required files</b>	<b>3</b>
<b>2</b>	<b>locale directory</b>	<b>3</b>
<b>3</b>	<b>Tools</b>	<b>3</b>
3.1	A few core definitions . . . . .	8
3.2	$\LaTeX$ : babel.sty (start) . . . . .	8
3.3	base . . . . .	9
3.4	key=value options and other general option . . . . .	10
3.5	Post-process some options . . . . .	11
3.6	Plain: babel.def (start) . . . . .	13
<b>4</b>	<b>babel.sty and babel.def (common)</b>	<b>13</b>
4.1	Selecting the language . . . . .	15
4.2	Errors . . . . .	23
4.3	More on selection . . . . .	23
4.4	Short tags . . . . .	25
4.5	Compatibility with language.def . . . . .	25
4.6	Hooks . . . . .	26
4.7	Setting up language files . . . . .	27
4.8	Shorthands . . . . .	29
4.9	Language attributes . . . . .	38
4.10	Support for saving and redefining macros . . . . .	39
4.11	French spacing . . . . .	40
4.12	Hyphens . . . . .	41
4.13	Multiencoding strings . . . . .	43
4.14	Tailor captions . . . . .	48
4.15	Making glyphs available . . . . .	49
4.15.1	Quotation marks . . . . .	49
4.15.2	Letters . . . . .	50
4.15.3	Shorthands for quotation marks . . . . .	51
4.15.4	Umlauts and tremas . . . . .	52
4.16	Layout . . . . .	53
4.17	Load engine specific macros . . . . .	54
4.18	Creating and modifying languages . . . . .	54
4.19	Main loop in ‘provide’ . . . . .	61
4.20	Processing keys in ini . . . . .	66
4.21	French spacing (again) . . . . .	71
4.22	Handle language system . . . . .	72
4.23	Numerals . . . . .	73
4.24	Casing . . . . .	74
4.25	Getting info . . . . .	75
4.26	BCP 47 related commands . . . . .	76
<b>5</b>	<b>Adjusting the Babel behavior</b>	<b>77</b>
5.1	Cross referencing macros . . . . .	79
5.2	Layout . . . . .	82
5.3	Marks . . . . .	83
5.4	Other packages . . . . .	84
5.4.1	ifthen . . . . .	84
5.4.2	varioref . . . . .	85
5.4.3	hhline . . . . .	85
5.5	Encoding and fonts . . . . .	86
5.6	Basic bidi support . . . . .	88
5.7	Local Language Configuration . . . . .	91
5.8	Language options . . . . .	91

<b>6</b>	<b>The kernel of Babel</b>	<b>95</b>
<b>7</b>	<b>Error messages</b>	<b>95</b>
<b>8</b>	<b>Loading hyphenation patterns</b>	<b>99</b>
<b>9</b>	<b>luatex + xetex: common stuff</b>	<b>103</b>
<b>10</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>106</b>
10.1	XeTeX . . . . .	106
10.2	Support for interchar . . . . .	108
10.3	Layout . . . . .	110
10.4	8-bit TeX . . . . .	111
10.5	LuaTeX . . . . .	112
10.6	Southeast Asian scripts . . . . .	119
10.7	CJK line breaking . . . . .	120
10.8	Arabic justification . . . . .	122
10.9	Common stuff . . . . .	126
10.10	Automatic fonts and ids switching . . . . .	127
10.11	Bidi . . . . .	133
10.12	Layout . . . . .	136
10.13	Lua: transforms . . . . .	145
10.14	Lua: Auto bidi with basic and basic-r . . . . .	155
<b>11</b>	<b>Data for CJK</b>	<b>167</b>
<b>12</b>	<b>The ‘nil’ language</b>	<b>167</b>
<b>13</b>	<b>Calendars</b>	<b>168</b>
13.1	Islamic . . . . .	168
13.2	Hebrew . . . . .	170
13.3	Persian . . . . .	174
13.4	Coptic and Ethiopic . . . . .	174
13.5	Buddhist . . . . .	175
<b>14</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>176</b>
14.1	Not renaming hyphen.tex . . . . .	176
14.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	177
14.3	General tools . . . . .	177
14.4	Encoding related macros . . . . .	181
<b>15</b>	<b>Acknowledgements</b>	<b>184</b>

The babel package is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use babel in real documents only as documented (except, of course, if you want to explore and test them).

## 1. Identification and loading of required files

The babel package after unpacking consists of the following files:

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

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

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

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

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

There some additional tex, def and lua files.

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

## 2. locale directory

A required component of babel is a set of ini files with basic definitions for about 300 languages. They are distributed as a separate zip file, not packed as dtx. Many of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (e.g., there are no geographic areas in Spanish). Not all include LICR variants.

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

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

## 3. Tools

```
1 <<version=25.8.86174>>
2 <<date=2025/05/10>>
```

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

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

**\bbl@afterfi** Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an \if-statement<sup>1</sup>. These macros will break if another \if... \fi statement appears in one of the arguments and it is not enclosed in braces.

```

31 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
32 \long\def\bbl@afterfi#1\fi{\fi#1}

```

**\bbl@exp** Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here \ stands for \noexpand, \< for \noexpand applied to a built macro name (which does not define the macro if undefined to \relax, because it is created locally), and \[. . .] for one-level expansion (where . . . is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```

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

```

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

```

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

```

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

**\bbl@ifunset** To check if a macro is defined, we create a new macro, which does the same as `\ifundefined`. However, in an  $\epsilon$ -tex engine, it is based on `\ifcurname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcurname` being implicitly set to `\relax` by the `\curname` test.

```

56 \begingroup
57 \gdef\bbl@ifunset#1{%
58   \expandafter\ifx\curname#1\endcurname\relax
59   \expandafter\@firstoftwo
60   \else
61   \expandafter\@secondoftwo
62   \fi}
63 \bbl@ifunset{ifcurname}%
64 {}%
65 {\gdef\bbl@ifunset#1{%
66   \ifcurname#1\endcurname
67   \expandafter\ifx\curname#1\endcurname\relax
68   \bbl@afterelse\expandafter\@firstoftwo
69   \else
70   \bbl@afterfi\expandafter\@secondoftwo
71   \fi
72   \else
73   \expandafter\@firstoftwo
74   \fi}}
75 \endgroup

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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



```

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

```

### 3.1. A few core definitions

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

```

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

```

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

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

```

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

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).

Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

### 3.3. base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that  $\LaTeX$  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.

```

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

```

### 3.4. key=value options and other general option

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

```

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

```

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

to use the shorthand characters in the preamble of their documents this can help.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

380 \ProcessOptions*

```

### 3.5. Post-process some options

```

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

```

390 \fi

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

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

First, exit immediately if previously loaded.

```

448 \ifx\ldf@quit\undefined\else
449 \endinput\fi % Same line!
450 \ifx\ldf@quit\defined\else
451 \ifx\ldf@quit\defined\else
452 \ifx\ldf@quit\defined\else
453 \ifx\ldf@quit\defined\else
454 \ifx\ldf@quit\defined\else
455 \ifx\ldf@quit\defined\else
456 \ifx\ldf@quit\defined\else
457 \ifx\ldf@quit\defined\else

```

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.

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

```

458 \ifx\ldf@quit\defined\else
459 \ifx\ldf@quit\defined\else
460 \ifx\ldf@quit\defined\else
461 \ifx\ldf@quit\defined\else
462 \ifx\ldf@quit\defined\else
463 \ifx\ldf@quit\defined\else
464 \ifx\ldf@quit\defined\else
465 \ifx\ldf@quit\defined\else
466 \ifx\ldf@quit\defined\else
467 \ifx\ldf@quit\defined\else
468 \ifx\ldf@quit\defined\else
469 \ifx\ldf@quit\defined\else
470 \ifx\ldf@quit\defined\else
471 \ifx\ldf@quit\defined\else
472 \ifx\ldf@quit\defined\else
473 \ifx\ldf@quit\defined\else
474 \ifx\ldf@quit\defined\else
475 \ifx\ldf@quit\defined\else
476 \ifx\ldf@quit\defined\else

```

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

```

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

```

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

The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s an attempt to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility

(perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `\l` is encapsulated, so that its case does not change.

```

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

```

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP 47 code.

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

```

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

```

```

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

```

**\iflanguage** Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, `\iflanguage`, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of `\language`. Then, depending on the result of the comparison, it executes either the second or the third argument.

```

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

```

## 4.1. Selecting the language

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

```

543 \let\bbl@select@type\z@
544 \edef\selectlanguage{%
545 \noexpand\protect
546 \expandafter\noexpand\csname selectlanguage \endcsname}

```

Because the command `\selectlanguage` could be used in a moving argument it expands to `\protect\selectlanguage_`. Therefore, we have to make sure that a macro `\protect` exists. If it doesn't it is `\let` to `\relax`.

```

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

```

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

```

548 \let\xstring\string

```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

**\bbl@pop@language** But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's `aftergroup` mechanism to help us. The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

**\bbl@language@stack** The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```

549 \def\bbl@language@stack{}

```

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

**\bbl@push@language**



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

```

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

```

Retrieving information from the stack is a little bit less simple, as we need to remove the element from the stack while storing it in the macro \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.

```

562 \def\bbl@pop@lang#1+#2\@@{%
563   \edef\language{#1}%
564   \xdef\bbl@language@stack{#2}}

```

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

```

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

```

Once the name of the previous language is retrieved from the stack, it is fed to \bbl@set@language to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of \localeid. This means \l@... will be reserved for hyphenation patterns (so that two locales can share the same rules).

```

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

```

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

```

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

```

```

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

```

**\bbl@set@language** The macro \bbl@set@language takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of \language. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in \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).

```

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

```

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

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

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

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\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 `\{language\}hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\{language\}hyphenmins` will be used.

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

```

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

```

```

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

```

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

```

733 \long\def\otherlanguage#1{%
734 \def\bbl@selectorname{other}%
735 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
736 \csname selectlanguage\endcsname{#1}%
737 \ignorespaces}

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

```

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

```

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

```

778 \def\foreign@language#1{%
779   % set name
780   \edef\language#1%
781   \ifbbl@usedategroup
782     \bbl@add\bbl@select@opts{,date,}%
783     \bbl@usedategroupfalse
784   \fi
785   \bbl@fixname\language
786   \let\localename\language
787   \bbl@provide@locale
788   \bbl@iflanguage\language{%
789     \let\bbl@select@type\@ne
790     \expandafter\bbl@switch\expandafter{\language}}

```

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

```

791 \def\IfBabelSelectorTF#1{%
792   \bbl@xin{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
793   \ifin@
794     \expandafter\@firstoftwo
795   \else
796     \expandafter\@secondoftwo
797   \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.

```

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

```

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

```

826 \def\hyphenrules#1{%
827   \edef\bbl@tempf{#1}%
828   \bbl@fixname\bbl@tempf
829   \bbl@iflanguage\bbl@tempf{%
830     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
831     \ifx\languageshorthands\@undefined\else
832       \languageshorthands{none}%
833     \fi
834     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
835       \set@hyphenmins\tw@\thr@\relax
836     \else
837       \expandafter\expandafter\expandafter\set@hyphenmins
838       \csname\bbl@tempf hyphenmins\endcsname\relax
839     \fi}}
840 \let\endhyphenrules\@empty

```

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

```

841 \def\providehyphenmins#1#2{%
842   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
843     \@namedef{#1hyphenmins}{#2}%
844   \fi}

```

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

```

845 \def\set@hyphenmins#1#2{%
846   \lefthyphenmin#1\relax
847   \righthyphenmin#2\relax}

```

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

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

```

848 \ifx\ProvidesFile\@undefined
849   \def\ProvidesLanguage#1[#2 #3 #4]{%
850     \wlog{Language: #1 #4 #3 <#2>}%
851   }
852 \else
853   \def\ProvidesLanguage#1{%
854     \begingroup
855       \catcode`\ 10 %
856       \@makeother\%
857       \@ifnextchar[%]
858         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
859   \def\@provideslanguage#1[#2]{%
860     \wlog{Language: #1 #2}%
861     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
862   \endgroup}
863 \fi

```

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

```

864 \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.

```

865 \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’:

```
866 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}}
867 \let\uselocale\setlocale
868 \let\locale\setlocale
869 \let\selectlocale\setlocale
870 \let\textlocale\setlocale
871 \let\textlanguage\setlocale
872 \let\languagegetext\setlocale
```

## 4.2. Errors

### **\@nolanerr**

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

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

When the format knows about \PackageError it must be L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

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

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

903 \ifx\bbl@onlyswitch\@empty\endinput\fi
```

## 4.3. More on selection

**\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.

```

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

```

```

960      \expandafter\bbl@tempa
961      \fi}%
962      \bbl@tempa#1\@empty}
963 \def\bbl@captionslist{%
964   \prefacename\refname\abstractname\bibname\chaptername\appendixname
965   \contentsname\listfigurename\listtablename\indexname\figurename
966   \tablename\partname\enclname\ccname\headtoname\pagename\seename
967   \alsoname\proofname\glossaryname}

```

#### 4.4. Short tags

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

```

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

```

#### 4.5. Compatibility with language.def

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

```

983 \bbl@trace{Compatibility with language.def}
984 \ifx\directlua\@undefined\else
985   \ifx\bbl@luapatterns\@undefined
986     \input luababel.def
987   \fi
988 \fi
989 \ifx\bbl@languages\@undefined
990   \ifx\directlua\@undefined
991     \openin1 = language.def
992     \ifeof1
993       \closein1
994       \message{I couldn't find the file language.def}
995     \else
996       \closein1
997       \begingroup
998         \def\addlanguage#1#2#3#4#5{%
999           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1000             \global\expandafter\let\csname l@#1\endcsname
1001               \csname lang@#1\endcsname
1002           \fi}%
1003         \def\uselanguage#1{%
1004           \input language.def
1005         \endgroup
1006       \fi
1007     \fi
1008     \chardef\l@english\z@
1009 \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.

```

1010 \def\addto#1#2{%
1011   \ifx#1\undefined
1012     \def#1{#2}%
1013   \else
1014     \ifx#1\relax
1015       \def#1{#2}%
1016     \else
1017       {\toks\expandafter{#1#2}%
1018        \xdef#1{\the\toks@}}%
1019   \fi
1020 \fi}

```

## 4.6. Hooks

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

```

1021 \bbl@trace{Hooks}
1022 \newcommand\AddBabelHook[3][]{%
1023   \bbl@iifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{%
1024     \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1025     \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1026     \bbl@iifunset{bbl@ev@#2@#3@#1}%
1027     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1028     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1029     \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1030 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1031 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1032 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1033 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1034   \ifx\UseHook\undefined\else\UseHook{babel/*/#2}\fi
1035   \def\bbl@elth##1{%
1036     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1037     \bbl@cs{ev@#2@#3}%
1038     \ifx\language\undefined\else % Test required for Plain (?)
1039       \ifx\UseHook\undefined\else\UseHook{babel/#1/#2}\fi
1040       \def\bbl@elth##1{%
1041         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}%
1042         \bbl@cs{ev@#2@#1}%
1043       \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).

```

1044 \def\bbl@evargs{,% <- don't delete this comma
1045   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1046   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1047   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1048   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1049   beforestart=0,language=2,begindocument=1}
1050 \ifx\NewHook\undefined\else % Test for Plain (?)
1051   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1052   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1053 \fi

```

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

```

1054 \providecommand\PassOptionsToLocale[2]{%
1055   \bbl@csarg\bbl@add@list{passto@#2}{#1}}

```

## 4.7. Setting up language files

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

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

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

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

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

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

Finally we check \originalTeX.

```
1056 \bbl@trace{Macros for setting language files up}
1057 \def\bbl@ldfinit{%
1058   \let\bbl@screset\@empty
1059   \let\BabelStrings\bbl@opt@string
1060   \let\BabelOptions\@empty
1061   \let\BabelLanguages\relax
1062   \ifx\originalTeX\@undefined
1063     \let\originalTeX\@empty
1064   \else
1065     \originalTeX
1066   \fi}
1067 \def\LdfInit#1#2{%
1068   \chardef\atcatcode=\catcode`\@
1069   \catcode`\@=11\relax
1070   \chardef\eqcatcode=\catcode`\=
1071   \catcode`\==12\relax
1072   \@ifpackagewith{babel}{ensureinfo=off}}}%
1073   {\ifx\InputIfFileExists\@undefined\else
1074     \bbl@ifunset\bbl@lname@#1}%
1075     {{\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1076       \def\languagename{#1}%
1077       \bbl@id@assign
1078       \bbl@load@info{#1}}}%
1079     {}%
1080   \fi}%
1081   \expandafter\if\expandafter\@backslashchar
1082     \expandafter\@car\string#2\@nil
1083   \ifx#2\@undefined\else
1084     \ldf@quit{#1}%
1085   \fi
1086 \else
1087   \expandafter\ifx\csname#2\endcsname\relax\else
1088     \ldf@quit{#1}%
1089   \fi
1090 \fi
1091 \bbl@ldfinit}
```

**\ldf@quit** This macro interrupts the processing of a language definition file. Remember \endinput is not executed immediately, but delayed to the end of the current line in the input file.

```
1092 \def\ldf@quit#1{%
1093   \expandafter\main@language\expandafter{#1}%
1094   \catcode`\@=\atcatcode \let\atcatcode\relax
```

```

1095 \catcode`\==\eqcatcode \let\eqcatcode\relax
1096 \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.

```

1097 \def\bbl@afterldf{%
1098 \bbl@afterlang
1099 \let\bbl@afterlang\relax
1100 \let\BabelModifiers\relax
1101 \let\bbl@screset\relax}%
1102 \def\ldf@finish#1{%
1103 \loadlocalcfg{#1}%
1104 \bbl@afterldf
1105 \expandafter\main@language\expandafter{#1}%
1106 \catcode`\@=\atcatcode \let\atcatcode\relax
1107 \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 *L<sup>A</sup>T<sub>E</sub>X*.

```

1108 \@onlypreamble\LdfInit
1109 \@onlypreamble\ldf@quit
1110 \@onlypreamble\ldf@finish

```

### **\main@language**

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

```

1111 \def\main@language#1{%
1112 \def\bbl@main@language{#1}%
1113 \let\language\name\bbl@main@language
1114 \let\localename\bbl@main@language
1115 \let\mainlocalename\bbl@main@language
1116 \bbl@id@assign
1117 \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.

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

```

1118 \def\bbl@beforestart{%
1119 \def\@nolanerr##1{%
1120 \bbl@carg\chardef{l@##1}\z@
1121 \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1122 \bbl@usehooks{beforestart}{}%
1123 \global\let\bbl@beforestart\relax}
1124 \AtBeginDocument{%
1125 {\@nameuse\bbl@beforestart}}% Group!
1126 \if@files
1127 \providecommand\babel@aux[2]{}%
1128 \immediate\write\@mainaux{\unexpanded{%
1129 \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1130 \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1131 \fi
1132 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1133 \ifbbl@single % must go after the line above.
1134 \renewcommand\selectlanguage[1]{}%
1135 \renewcommand\foreignlanguage[2]{#2}%
1136 \global\let\babel@aux\@gobbletwo % Also as flag
1137 \fi}

```

```

1138 %
1139 \ifcase\bbl@engine\or
1140   \AtBeginDocument{\pagedir\bodydir}
1141 \fi

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

1142 \def\select@language@x#1{%
1143   \ifcase\bbl@select@type
1144     \bbl@ifsamestring\language#1\{\select@language{#1}}%
1145   \else
1146     \select@language{#1}%
1147   \fi}

```

## 4.8. Shorthands

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

```

1148 \bbl@trace{Shorhands}
1149 \def\bbl@withactive#1#2{%
1150   \begingroup
1151     \lccode`~=#2\relax
1152     \lowercase{\endgroup#1~}}

```

**\bbl@add@special** The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if  $\TeX$  is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1153 \def\bbl@add@special#1{% 1:a macro like "\", \?, etc.
1154   \bbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1155   \bbl@ifunset{\@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1156   \ifx\nfss@catcodes\undefined\else
1157     \begingroup
1158       \catcode`#1\active
1159       \nfss@catcodes
1160       \ifnum\catcode`#1=\active
1161         \endgroup
1162         \bbl@add\nfss@catcodes{\@makeother#1}%
1163       \else
1164         \endgroup
1165       \fi
1166   \fi}

```

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

For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines `"` as `\active@prefix "\active@char` (where the first `"` is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (i.e., with the original `"`); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (e.g., `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` 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).

```

1167 \def\bbl@active@def#1#2#3#4{%
1168   \@namedef{#3#1}{%
1169     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1170       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1171     \else
1172       \bbl@afterfi\csname#2@sh@#1@\endcsname
1173     \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.

```

1174   \long\@namedef{#3@arg#1}##1{%
1175     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1176       \bbl@afterelse\csname#4#1\endcsname##1%
1177     \else
1178       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1179     \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.

```

1180 \def\initiate@active@char#1{%
1181   \bbl@ifunset{active@char\string#1}%
1182   {\bbl@withactive
1183     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1184   {}}

```

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).

```

1185 \def\@initiate@active@char#1#2#3{%
1186   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1187   \ifx#1\@undefined
1188     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1189   \else
1190     \bbl@csarg\let{oridef@#2}#1%
1191     \bbl@csarg\edef{oridef@#2}{%
1192       \let\noexpand#1%
1193       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1194   \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*").

```

1195   \ifx#1#3\relax
1196     \expandafter\let\csname normal@char#2\endcsname#3%
1197   \else
1198     \bbl@info{Making #2 an active character}%
1199     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1200     \@namedef{normal@char#2}{%
1201       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1202     \else
1203       \@namedef{normal@char#2}{#3}%
1204     \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).

```

1205   \bbl@restoreactive{#2}%
1206   \AtBeginDocument{%

```

```

1207 \catcode`#2\active
1208 \if@filesw
1209 \immediate\write\@mainaux{\catcode`\string#2\active}%
1210 \fi}%
1211 \expandafter\bbbl@add@special\csname#2\endcsname
1212 \catcode`#2\active
1213 \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⟩`).

```

1214 \let\bbbl@tempa\@firstoftwo
1215 \if\string^#2%
1216 \def\bbbl@tempa{\noexpand\textormath}%
1217 \else
1218 \ifx\bbbl@mathnormal\@undefined\else
1219 \let\bbbl@tempa\bbbl@mathnormal
1220 \fi
1221 \fi
1222 \expandafter\edef\csname active@char#2\endcsname{%
1223 \bbbl@tempa
1224 {\noexpand\if@safe@actives
1225 \noexpand\expandafter
1226 \expandafter\noexpand\csname normal@char#2\endcsname
1227 \noexpand\else
1228 \noexpand\expandafter
1229 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1230 \noexpand\fi}%
1231 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1232 \bbbl@csarg\edef{doactive#2}{%
1233 \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

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

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

```

1234 \bbbl@csarg\edef{active@#2}{%
1235 \noexpand\active@prefix\noexpand#1%
1236 \expandafter\noexpand\csname active@char#2\endcsname}%
1237 \bbbl@csarg\edef{normal@#2}{%
1238 \noexpand\active@prefix\noexpand#1%
1239 \expandafter\noexpand\csname normal@char#2\endcsname}%
1240 \bbbl@ncarg\let#1\bbbl@normal@#2}%

```

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn’t exist we check for a shorthand with an argument.

```

1241 \bbbl@active@def#2\user@group{user@active}{language@active}%
1242 \bbbl@active@def#2\language@group{language@active}{system@active}%
1243 \bbbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1244 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1245 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1246 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1247 {\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.

```
1248 \if\string'#2%
1249 \let\prim@s\bbl@prim@s
1250 \let\active@math@prime#1%
1251 \fi
1252 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
1253 <<*More package options>> ≡
1254 \DeclareOption{math=active}{}
1255 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1256 <</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*.

```
1257 \ifpackagewith{babel}{KeepShorthandsActive}%
1258 {\let\bbl@restoreactive\@gobble}%
1259 {\def\bbl@restoreactive#1{%
1260 \bbl@exp{%
1261 \\\AfterBabelLanguage\\CurrentOption
1262 {\catcode`#1=\the\catcode`#1\relax}%
1263 \\\AtEndOfPackage
1264 {\catcode`#1=\the\catcode`#1\relax}}}%
1265 \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.

```
1266 \def\bbl@sh@select#1#2{%
1267 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1268 \bbl@afterelse\bbl@scndcs
1269 \else
1270 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1271 \fi}
```

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

```
1272 \begingroup
1273 \bbl@ifunset{ifincsname}
1274 {\gdef\active@prefix#1{%
1275 \ifx\protect\@typeset@protect
1276 \else
1277 \ifx\protect\@unexpandable@protect
1278 \noexpand#1%
1279 \else
1280 \protect#1%
1281 \fi
1282 \expandafter\@gobble
1283 \fi}}
1284 {\gdef\active@prefix#1{%
1285 \ifincsname
```

```

1286     \string#1%
1287     \expandafter\@gobble
1288   \else
1289     \ifx\protect\@typeset@protect
1290     \else
1291       \ifx\protect\@unexpandable@protect
1292         \noexpand#1%
1293       \else
1294         \protect#1%
1295       \fi
1296     \expandafter\expandafter\expandafter\@gobble
1297   \fi
1298 \fi}}
1299 \endgroup

```

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

```

1300 \newif\if@safe@actives
1301 \@safe@activesfalse

```

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

```

1302 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

**\bbl@activate**

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

```

1303 \chardef\bbl@activated\z@
1304 \def\bbl@activate#1{%
1305   \chardef\bbl@activated\@ne
1306   \bbl@withactive{\expandafter\let\expandafter}#1%
1307   \csname bbl@active@\string#1\endcsname}
1308 \def\bbl@deactivate#1{%
1309   \chardef\bbl@activated\tw@
1310   \bbl@withactive{\expandafter\let\expandafter}#1%
1311   \csname bbl@normal@\string#1\endcsname}

```

**\bbl@firstcs**

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

```

1312 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1313 \def\bbl@scndcs#1#2{\csname#2\endcsname}

```

**\declare@shorthand** Used to declare a shorthand on a certain level. It takes three arguments:

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

The auxiliary macro \babel@texpdf improves the interoperativity with hyperref and takes 4 arguments: (1) The T<sub>E</sub>X code in text mode, (2) the string for hyperref, (3) the T<sub>E</sub>X code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently hyperref doesn’t discriminate the mode). This macro may be used in l<sub>at</sub>e<sub>x</sub> files.

```

1314 \def\babel@texpdf#1#2#3#4{%

```

```

1315 \ifx\texorpdfstring\undefined
1316   \textormath{#1}{#3}%
1317 \else
1318   \texorpdfstring{\textormath{#1}{#3}}{#2}%
1319   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1320 \fi}
1321 %
1322 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1323 \def\@decl@short#1#2#3\@nil#4{%
1324   \def\bbl@tempa{#3}%
1325   \ifx\bbl@tempa\@empty
1326     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1327     \bbl@ifunset{#1@sh@\string#2@}{}%
1328     {\def\bbl@tempa{#4}%
1329       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1330       \else
1331         \bbl@info
1332           {Redefining #1 shorthand \string#2\\%
1333            in language \CurrentOption}%
1334       \fi}%
1335     \@namedef{#1@sh@\string#2@}{#4}%
1336   \else
1337     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1338     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1339     {\def\bbl@tempa{#4}%
1340       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1341       \else
1342         \bbl@info
1343           {Redefining #1 shorthand \string#2\string#3\\%
1344            in language \CurrentOption}%
1345       \fi}%
1346     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1347   \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.

```

1348 \def\textormath{%
1349   \ifmmode
1350     \expandafter\@secondoftwo
1351   \else
1352     \expandafter\@firstoftwo
1353   \fi}

```

**\user@group**

**\language@group**

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

```

1354 \def\user@group{user}
1355 \def\language@group{english}
1356 \def\system@group{system}

```

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

```

1357 \def\useshorthands{%
1358   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1359 \def\bbl@usesh@s#1{%
1360   \bbl@usesh@x
1361   {\AddBabelHook{babel-sh-\string#1}{afterextras}}{\bbl@activate{#1}}}%
1362   {#1}}

```

```

1363 \def\bbl@usesh@x#1#2{%
1364   \bbl@ifshorthand{#2}%
1365   {\def\user@group{user}%
1366    \initiate@active@char{#2}%
1367    #1%
1368    \bbl@activate{#2}}%
1369   {\bbl@error{shorthand-is-off}{#2}{}}}

```

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

```

1370 \def\user@language@group{user@\language@group}
1371 \def\bbl@set@user@generic#1#2{%
1372   \bbl@ifunset{user@generic@active#1}%
1373   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1374    \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1375    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1376     \expandafter\noexpand\csname normal@char#1\endcsname}%
1377    \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1378     \expandafter\noexpand\csname user@active#1\endcsname}%
1379    \@empty}
1380 \newcommand\defineshorthand[3][user]{%
1381   \edef\bbl@tempa{\zap@space#1 \@empty}%
1382   \bbl@for\bbl@tempb\bbl@tempa{%
1383     \if*\expandafter\@car\bbl@tempb\@nil
1384     \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1385     \@expandtwoargs
1386     \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1387   \fi
1388   \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

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

```

1389 \def\languageshorthands#1{%
1390   \bbl@ifsamestring{none}{#1}{}%
1391   \bbl@once{short-\localename-#1}{%
1392     \bbl@info{'\localename' activates '#1' shorthands.\Reported }}%
1393   \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`.

```

1394 \def\aliasshorthand#1#2{%
1395   \bbl@ifshorthand{#2}%
1396   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1397     \ifx\document@notprerr
1398       \@notshorthand{#2}%
1399     \else
1400       \initiate@active@char{#2}%
1401       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1402       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1403       \bbl@activate{#2}%
1404     \fi
1405   \fi}%
1406   {\bbl@error{shorthand-is-off}{#2}{}}}

```

**\@notshorthand**

```

1407 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}

```

## **\shorthandon**

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

```
1408 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1409 \DeclareRobustCommand*\shorthandoff{%
1410   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1411 \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.

```
1412 \def\bbl@switch@sh#1#2{%
1413   \ifx#2\@nnil\else
1414     \bbl@ifunset{bbl@active@\string#2}%
1415     {\bbl@error{not-a-shorthand-b}{\string#2}}}%
1416     {\ifcase#1%   off, on, off*
1417       \catcode`#2\relax
1418       \or
1419       \catcode`#2\active
1420       \bbl@ifunset{bbl@shdef@\string#2}%
1421       {}%
1422       {\bbl@withactive{\expandafter\let\expandafter}#2%
1423         \csname bbl@shdef@\string#2\endcsname
1424         \bbl@csarg\let{shdef@\string#2}\relax}%
1425       \ifcase\bbl@activated\or
1426       \bbl@activate{#2}%
1427       \else
1428       \bbl@deactivate{#2}%
1429       \fi
1430       \or
1431       \bbl@ifunset{bbl@shdef@\string#2}%
1432       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1433       {}%
1434       \csname bbl@oricat@\string#2\endcsname
1435       \csname bbl@oridef@\string#2\endcsname
1436       \fi}%
1437   \bbl@afterfi\bbl@switch@sh#1%
1438   \fi}
```

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

```
1439 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1440 \def\bbl@putsh#1{%
1441   \bbl@ifunset{bbl@active@\string#1}%
1442   {\bbl@putsh@i#1\@empty\@nnil}%
1443   {\csname bbl@active@\string#1\endcsname}}
1444 \def\bbl@putsh@i#1#2\@nnil{%
1445   \csname\language@group @sh@\string#1@%
1446     \ifx\@empty#2\else\string#2@\fi\endcsname}
1447 %
1448 \ifx\bbl@opt@shorthands\@nnil\else
1449   \let\bbl@s@initiate@active@char\initiate@active@char
1450   \def\initiate@active@char#1{%
1451     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1452   \let\bbl@s@switch@sh\bbl@switch@sh
1453   \def\bbl@switch@sh#1#2{%
1454     \ifx#2\@nnil\else
```

```

1455 \bbl@afterfi
1456 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1457 \fi}
1458 \let\bbl@s@activate\bbl@activate
1459 \def\bbl@activate#1{%
1460 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1461 \let\bbl@s@deactivate\bbl@deactivate
1462 \def\bbl@deactivate#1{%
1463 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1464 \fi

```

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

```

1465 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@string#1}{#3}{#2}}

```

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

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

```

1466 \def\bbl@prim@s{%
1467 \prime\futurelet\@let@token\bbl@pr@m@s}
1468 \def\bbl@if@primes#1#2{%
1469 \ifx#1\@let@token
1470 \expandafter\@firstoftwo
1471 \else\ifx#2\@let@token
1472 \bbl@afterelse\expandafter\@firstoftwo
1473 \else
1474 \bbl@afterfi\expandafter\@secondoftwo
1475 \fi\fi}
1476 \begingroup
1477 \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1478 \catcode`\'=12 \catcode`\"=\active \lccode`\"='\ '
1479 \lowercase{%
1480 \gdef\bbl@pr@m@s{%
1481 \bbl@if@primes" '%
1482 \pr@@@s
1483 {\bbl@if@primes*\^pr@@@t\egroup}}}
1484 \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).

```

1485 \initiate@active@char{~}
1486 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1487 \bbl@activate{~}

```

## **\OT1dqpos**

**\T1dqpos** The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the `\f@encoding` macro. Therefore we define two macros here to store the position of the character in these encodings.

```

1488 \expandafter\def\csname OT1dqpos\endcsname{127}
1489 \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

```

1490 \ifx\f@encoding\undefined
1491 \def\f@encoding{OT1}
1492 \fi

```

## 4.9. Language attributes

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

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

```
1493 \bbl@trace{Language attributes}
1494 \newcommand\languageattribute[2]{%
1495   \def\bbl@tempc{#1}%
1496   \bbl@fixname\bbl@tempc
1497   \bbl@iflanguage\bbl@tempc{%
1498     \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.

```
1499     \ifx\bbl@known@attrs\undefined
1500       \in@false
1501     \else
1502       \bbl@xin{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1503     \fi
1504     \ifin@
1505       \bbl@warning{%
1506         You have more than once selected the attribute '##1'\%
1507         for language #1. Reported}%
1508     \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.

```
1509       \bbl@exp{%
1510         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1511       \edef\bbl@tempa{\bbl@tempc-##1}%
1512       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1513       {\csname\bbl@tempc @attr##1\endcsname}%
1514       {\@attrerr{\bbl@tempc}{##1}}%
1515     \fi}}
1516 \@onlypreamble\languageattribute
```

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

```
1517 \newcommand*\@attrerr[2]{%
1518   \bbl@error{unknown-attribute}{#1}{#2}{}}
```

**\bbl@declare@ttribute** This command adds the new language/attribute combination to the list of known attributes.

Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro `\extras...` for the current language is extended, otherwise the attribute will not work as its code is removed from memory at `\begin{document}`.

```
1519 \def\bbl@declare@ttribute#1#2#3{%
1520   \bbl@xin{,#2,}{,\BabelModifiers,}%
1521   \ifin@
1522     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1523   \fi
1524   \bbl@add@list\bbl@attributes{#1-#2}%
1525   \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.

```

1526 \def\bbl@ifattributeset#1#2#3#4{%
1527   \ifx\bbl@known@attrs\@undefined
1528     \in@false
1529   \else
1530     \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1531   \fi
1532   \ifin@
1533     \bbl@afterelse#3%
1534   \else
1535     \bbl@afterfi#4%
1536   \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.

```

1537 \def\bbl@ifknown@ttrib#1#2{%
1538   \let\bbl@tempa\@secondoftwo
1539   \bbl@loopx\bbl@tempb{#2}{%
1540     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1541   \ifin@
1542     \let\bbl@tempa\@firstoftwo
1543   \else
1544     \fi}%
1545   \bbl@tempa}

```

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

```

1546 \def\bbl@clear@ttribs{%
1547   \ifx\bbl@attributes\@undefined\else
1548     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1549       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1550     \let\bbl@attributes\@undefined
1551   \fi}
1552 \def\bbl@clear@ttrib#1-#2.{%
1553   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1554 \AtBeginDocument{\bbl@clear@ttribs}

```

## 4.10. Support for saving and redefining macros

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

**\babel@savecnt**

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

```

1555 \bbl@trace{Macros for saving definitions}
1556 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1557 \newcount\babel@savecnt
1558 \babel@beginsave

```

**\babel@save**



**\babel@savevariable** The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX` (which has to be expandable, i.e., you shouldn't let it to `\relax`). To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive. To avoid messing saved definitions up, they are saved only the very first time.

```

1559 \def\babel@save#1{%
1560   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1561   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1562     \expandafter{\expandafter,\bbl@savedextras,}}%
1563   \expandafter\in@\bbl@tempa
1564   \ifin@%else
1565     \bbl@add\bbl@savedextras{,{#1,}}%
1566     \bbl@carg\let\babel@number\babel@savecnt#1\relax
1567     \toks@{\expandafter{\originalTeX\let#1=}}%
1568     \bbl@exp{%
1569       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1570     \advance\babel@savecnt@one
1571   \fi}
1572 \def\babel@savevariable#1{%
1573   \toks@{\expandafter{\originalTeX #1=}}%
1574   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```

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

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

```

1580 \def\bbl@redefine@long#1{%
1581   \edef\bbl@tempa{\bbl@stripslash#1}%
1582   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1583   \long\expandafter\def\csname\bbl@tempa\endcsname}
1584 \@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_`.

```

1585 \def\bbl@redefineroobust#1{%
1586   \edef\bbl@tempa{\bbl@stripslash#1}%
1587   \bbl@ifunset{\bbl@tempa\space}%
1588     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1589       \bbl@exp{\def\\#1\\\\protect<\<\bbl@tempa\space>}}}%
1590     {\bbl@exp{\let<org@\bbl@tempa><\<\bbl@tempa\space>}}}%
1591     \@namedef{\bbl@tempa\space}}
1592 \@onlypreamble\bbl@redefineroobust
```

## 4.11. French spacing

**\bbl@frenchspacing**

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

```

1593 \def\bbl@frenchspacing{%
1594   \ifnum\the\sfcodes`\.=\@m
1595     \let\bbl@nonfrenchspacing\relax
1596   \else
1597     \frenchspacing
1598     \let\bbl@nonfrenchspacing\nonfrenchspacing
1599   \fi}
1600 \let\bbl@nonfrenchspacing\nonfrenchspacing

```

A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in \babelprovide. This new method should be ideally the default one.

```

1601 \let\bbl@elt\relax
1602 \edef\bbl@fs@chars{%
1603   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1604   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1605   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1606 \def\bbl@pre@fs{%
1607   \def\bbl@elt##1##2##3{\sfcodes`##1=\the\sfcodes`##1\relax}%
1608   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1609 \def\bbl@post@fs{%
1610   \bbl@save@sfcodes
1611   \edef\bbl@tempa{\bbl@cl{frspc}}%
1612   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1613   \if u\bbl@tempa      % do nothing
1614   \else\if n\bbl@tempa % non french
1615     \def\bbl@elt##1##2##3{%
1616       \ifnum\sfcodes`##1=##2\relax
1617       \babel@savevariable{\sfcodes`##1}%
1618       \sfcodes`##1=##3\relax
1619     \fi}%
1620     \bbl@fs@chars
1621   \else\if y\bbl@tempa % french
1622     \def\bbl@elt##1##2##3{%
1623       \ifnum\sfcodes`##1=##3\relax
1624       \babel@savevariable{\sfcodes`##1}%
1625       \sfcodes`##1=##2\relax
1626     \fi}%
1627     \bbl@fs@chars
1628   \fi\fi\fi}

```

## 4.12. Hyphens

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

```

1629 \bbl@trace{Hyphens}
1630 \@onlypreamble\babelhyphenation
1631 \AtEndOfPackage{%
1632   \newcommand\babelhyphenation[2][\@empty]{%
1633     \ifx\bbl@hyphenation@\relax
1634       \let\bbl@hyphenation@\@empty
1635     \fi
1636     \ifx\bbl@hyphlist\@empty\else
1637       \bbl@warning{%
1638         You must not intermingle \string\selectlanguage\space and\\%
1639         \string\babelhyphenation\space or some exceptions will not\\%
1640         be taken into account. Reported}%
1641     \fi

```

```

1642 \ifx\@empty#1%
1643 \protected@edef\bb@hyphenation@\bb@hyphenation\space#2}%
1644 \else
1645 \bb@vforeach{#1}{%
1646 \def\bb@tempa{##1}%
1647 \bb@fixname\bb@tempa
1648 \bb@iflanguage\bb@tempa{%
1649 \bb@csarg\protected@edef\hyphenation@\bb@tempa}{%
1650 \bb@ifunset\bb@hyphenation@\bb@tempa}%
1651 }%
1652 {\csname \bb@hyphenation@\bb@tempa\endcsname\space}%
1653 #2}}}%
1654 \fi}}

```

**\babelhyphenmins** Only L<sup>A</sup>T<sub>E</sub>X (basically because it's defined with a L<sup>A</sup>T<sub>E</sub>X tool).

```

1655 \ifx\NewDocumentCommand\@undefined\else
1656 \NewDocumentCommand\babelhyphenmins{sommo}{%
1657 \IfNoValueTF{#2}%
1658 {\protected@edef\bb@hyphenmins@\set@hyphenmins{#3}{#4}}%
1659 \IfValueT{#5}{%
1660 \protected@edef\bb@hyphenatmin@\hyphenationmin=#5\relax}}%
1661 \IfBooleanT{#1}{%
1662 \lefthyphenmin=#3\relax
1663 \righthyphenmin=#4\relax
1664 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1665 {\edef\bb@tempb{\zap@space#2 \@empty}%
1666 \bb@for\bb@tempa\bb@tempb{%
1667 \@namedef\bb@hyphenmins@\bb@tempa}{\set@hyphenmins{#3}{#4}}%
1668 \IfValueT{#5}{%
1669 \@namedef\bb@hyphenatmin@\bb@tempa}{\hyphenationmin=#5\relax}}}%
1670 \IfBooleanT{#1}{\bb@error{hyphenmins-args}{}}}%
1671 \fi

```

**\bb@allowhyphens** This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak\hskip 0pt plus 0pt`. T<sub>E</sub>X begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

1672 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1673 \def\bb@t@one{Tl}
1674 \def\allowhyphens{\ifx\cf@encoding\bb@t@one\else\bb@allowhyphens\fi}

```

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

```

1675 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1676 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1677 \def\bb@hyphen{%
1678 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
1679 \def\bb@hyphen@i#1#2{%
1680 \lowercase{\bb@ifunset\bb@hy@#1#2\@empty}}%
1681 {\csname \bb@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1682 {\lowercase{\csname \bb@hy@#1#2\@empty\endcsname}}}

```

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

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

```

1683 \def\bb@usehyphen#1{%
1684 \leavevmode

```

```

1685 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1686 \nobreak\hskip\z@skip}
1687 \def\bbl@usehyphen#1{%
1688 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1689 \def\bbl@hyphenchar{%
1690 \ifnum\hyphenchar\font=\m@ne
1691 \babelnullhyphen
1692 \else
1693 \char\hyphenchar\font
1694 \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.

```

1695 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1696 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1697 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1698 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1699 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1700 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1701 \def\bbl@hy@repeat{%
1702 \bbl@usehyphen{
1703 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1704 \def\bbl@hy@@repeat{%
1705 \bbl@usehyphen{
1706 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1707 \def\bbl@hy@empty{\hskip\z@skip}
1708 \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.

```

1709 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

```

## 4.13. Multiencoding strings

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

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

```

1710 \bbl@trace{Multiencoding strings}
1711 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1712 <<*More package options>> ≡
1713 \DeclareOption{nocase}{}
1714 <</More package options>>

```

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

```

1715 <<*More package options>> ≡
1716 \let\bbl@opt@strings\@nnil % accept strings=value
1717 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1718 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1719 \def\BabelStringsDefault{generic}
1720 <</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.

```

1721 \@onlypreamble\StartBabelCommands
1722 \def\StartBabelCommands{%
1723   \begingroup
1724   \@tempcnta="7F
1725   \def\bbl@tempa{%
1726     \ifnum\@tempcnta>"FF\else
1727       \catcode\@tempcnta=11
1728       \advance\@tempcnta\@ne
1729       \expandafter\bbl@tempa
1730     \fi}%
1731   \bbl@tempa
1732   <@Macros local to BabelCommands@>
1733   \def\bbl@provstring##1##2{%
1734     \providecommand##1{##2}%
1735     \bbl@tglobal##1}%
1736   \global\let\bbl@scafter\@empty
1737   \let\StartBabelCommands\bbl@startcmds
1738   \ifx\BabelLanguages\relax
1739     \let\BabelLanguages\CurrentOption
1740   \fi
1741   \begingroup
1742   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1743   \StartBabelCommands}
1744 \def\bbl@startcmds{%
1745   \ifx\bbl@screset\@nnil\else
1746     \bbl@usehooks{stopcommands}{}%
1747   \fi
1748   \endgroup
1749   \begingroup
1750   \@ifstar
1751     {\ifx\bbl@opt@strings\@nnil
1752       \let\bbl@opt@strings\BabelStringsDefault
1753     \fi
1754     \bbl@startcmds@i}%
1755   \bbl@startcmds@i}
1756 \def\bbl@startcmds@i##1##2{%
1757   \edef\bbl@L{\zap@space#1 \@empty}%
1758   \edef\bbl@G{\zap@space#2 \@empty}%
1759   \bbl@startcmds@ii}
1760 \let\bbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (i.e., fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (i.e., no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```

1761 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1762   \let\SetString\@gobbletwo
1763   \let\bbl@stringdef\@gobbletwo
1764   \let\AfterBabelCommands\@gobble
1765   \ifx\@empty#1%
1766     \def\bbl@sc@label{generic}%
1767     \def\bbl@encstring##1##2{%
1768       \ProvideTextCommandDefault##1{##2}%
1769       \bbl@tglobal##1%
1770       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1771 \let\bbl@sctest\in@true
1772 \else
1773 \let\bbl@sc@charset\space % <- zapped below
1774 \let\bbl@sc@fontenc\space % <- " "
1775 \def\bbl@tempa##1=##2\@nil{%
1776 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1777 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1778 \def\bbl@tempa##1 ##2{% space -> comma
1779 ##1%
1780 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1781 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1782 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1783 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1784 \def\bbl@encstring##1##2{%
1785 \bbl@foreach\bbl@sc@fontenc{%
1786 \bbl@ifunset{T@####1}%
1787 {}%
1788 {\ProvideTextCommand##1{####1}{##2}%
1789 \bbl@tglobal##1%
1790 \expandafter
1791 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1792 \def\bbl@sctest{%
1793 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1794 \fi
1795 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1796 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1797 \let\AfterBabelCommands\bbl@aftercmds
1798 \let\SetString\bbl@setstring
1799 \let\bbl@stringdef\bbl@encstring
1800 \else % i.e., strings=value
1801 \bbl@sctest
1802 \ifin@
1803 \let\AfterBabelCommands\bbl@aftercmds
1804 \let\SetString\bbl@setstring
1805 \let\bbl@stringdef\bbl@provstring
1806 \fi\fi\fi
1807 \bbl@scswitch
1808 \ifx\bbl@G\@empty
1809 \def\SetString##1##2{%
1810 \bbl@error{missing-group}{##1}{}}}%
1811 \fi
1812 \ifx\@empty#1%
1813 \bbl@usehooks{defaultcommands}{}%
1814 \else
1815 \@expandtwoargs
1816 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1817 \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).

```

1818 \def\bbl@forlang#1#2{%
1819 \bbl@for#1\bbl@L{%
1820 \bbl@xin@{,##1,}{,\BabelLanguages,}%
1821 \ifin@#2\relax\fi}}
1822 \def\bbl@scswitch{%
1823 \bbl@forlang\bbl@tempa{%
1824 \ifx\bbl@G\@empty\else

```

```

1825 \ifx\SetString@gobbletwo\else
1826 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1827 \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1828 \ifin@else
1829 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1830 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1831 \fi
1832 \fi
1833 \fi}}
1834 \AtEndOfPackage{%
1835 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}}{#2}}%
1836 \let\bbl@scswitch\relax}
1837 \@onlypreamble\EndBabelCommands
1838 \def\EndBabelCommands{%
1839 \bbl@usehooks{stopcommands}}}%
1840 \endgroup
1841 \endgroup
1842 \bbl@scafter}
1843 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

**Strings** The following macro is the actual definition of \SetString when it is “active”

First save the “switcher”. Create it if undefined. Strings are defined only if undefined (i.e., like \providescommand). With the event stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

1844 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1845 \bbl@forlang\bbl@tempa{%
1846 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1847 \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1848 {\bbl@exp{%
1849 \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1850 }%
1851 \def\BabelString{#2}%
1852 \bbl@usehooks{stringprocess}}}%
1853 \expandafter\bbl@stringdef
1854 \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```

1855 \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.

```

1856 << *Macros local to BabelCommands >> ≡
1857 \def\SetStringLoop##1##2{%
1858 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1859 \count@\z@
1860 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1861 \advance\count@\@ne
1862 \toks@\expandafter{\bbl@tempa}%
1863 \bbl@exp{%
1864 \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1865 \count@=\the\count@\relax}}}%
1866 << /Macros local to BabelCommands >>

```

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

```

1867 \def\bbl@aftercmds#1{%
1868 \toks@\expandafter{\bbl@scafter#1}%
1869 \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.

```

1870 <<*Macros local to BabelCommands>> ≡
1871   \newcommand\SetCase[3][]{%
1872     \def\bbl@tempa####1####2{%
1873       \ifx####1\@empty\else
1874         \bbl@carg\bbl@add{extras\CurrentOption}{%
1875           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1876           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1877           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1878           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1879       \expandafter\bbl@tempa
1880     \fi}%
1881   \bbl@tempa##1\@empty\@empty
1882   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1883 <</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.

```

1884 <<*Macros local to BabelCommands>> ≡
1885   \newcommand\SetHyphenMap[1]{%
1886     \bbl@forlang\bbl@tempa{%
1887       \expandafter\bbl@stringdef
1888       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1889 <</Macros local to BabelCommands>>

```

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

```

1890 \newcommand\BabelLower[2]{% one to one.
1891   \ifnum\lccode#1=#2\else
1892     \babel@savevariable{\lccode#1}%
1893     \lccode#1=#2\relax
1894   \fi}
1895 \newcommand\BabelLowerMM[4]{% many-to-many
1896   \@tempcnta=#1\relax
1897   \@tempcntb=#4\relax
1898   \def\bbl@tempa{%
1899     \ifnum\@tempcnta>#2\else
1900       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1901       \advance\@tempcnta#3\relax
1902       \advance\@tempcntb#3\relax
1903       \expandafter\bbl@tempa
1904     \fi}%
1905   \bbl@tempa}
1906 \newcommand\BabelLowerM0[4]{% many-to-one
1907   \@tempcnta=#1\relax
1908   \def\bbl@tempa{%
1909     \ifnum\@tempcnta>#2\else
1910       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1911       \advance\@tempcnta#3
1912       \expandafter\bbl@tempa
1913     \fi}%
1914   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1915 <<*More package options>> ≡
1916 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1917 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1918 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1919 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1920 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1921 <</More package options>>

```



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

```

1922 \AtEndOfPackage{%
1923   \ifx\bbbl@opt@hyphenmap\@undefined
1924     \bbbl@xin@{,}\bbbl@language@opts}%
1925     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1926   \fi}

```

## 4.14. Tailor captions

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

```

1927 \newcommand\setlocalecaption{%
1928   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1929 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1930   \bbbl@trim@def\bbbl@tempa{#2}%
1931   \bbbl@xin@{.template}\bbbl@tempa}%
1932   \ifin@
1933     \bbbl@ini@captions@template{#3}{#1}%
1934   \else
1935     \edef\bbbl@tempd{%
1936       \expandafter\expandafter\expandafter
1937       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1938     \bbbl@xin@
1939       {\expandafter\string\csname #2name\endcsname}%
1940     {\bbbl@tempd}%
1941     \ifin@ % Renew caption
1942       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
1943     \ifin@
1944       \bbbl@exp{%
1945         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1946         {\bbbl@scset\<#2name>\<#1#2name>}%
1947         {}}%
1948       \else % Old way converts to new way
1949         \bbbl@ifunset{#1#2name}%
1950         {\bbbl@exp{%
1951           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1952           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1953           {\def\<#2name>\<#1#2name>}}%
1954         {}}}%
1955       {}%
1956     \fi
1957   \else
1958     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
1959     \ifin@ % New way
1960       \bbbl@exp{%
1961         \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
1962         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1963         {\bbbl@scset\<#2name>\<#1#2name>}%
1964         {}}%
1965       \else % Old way, but defined in the new way
1966         \bbbl@exp{%
1967           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1968           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1969           {\def\<#2name>\<#1#2name>}}%
1970         {}}%
1971       \fi%
1972     \fi
1973     \@namedef{#1#2name}{#3}%
1974     \toks@ \expandafter\bbbl@captionslist}%
1975     \bbbl@exp{\in@{\<#2name>}\the\toks@}%
1976     \ifin@ \else
1977       \bbbl@exp{\bbbl@add\bbbl@captionslist{\<#2name>}}%

```

```

1978      \bbl@tglobal\bbl@captionslist
1979      \fi
1980      \fi}

```

## 4.15. Making glyphs available

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

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

```

1981 \bbl@trace{Macros related to glyphs}
1982 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1983      \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
1984      \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.

```

1985 \def\save@sf@q#1{\leavevmode
1986      \begingroup
1987      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1988      \endgroup}

```

### 4.15.1. Quotation marks

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

```

1989 \ProvideTextCommand{\quotedblbase}{OT1}{%
1990      \save@sf@q{\set@low@box{\textquotedblright/}}%
1991      \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.

```

1992 \ProvideTextCommandDefault{\quotedblbase}{%
1993      \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1994 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1995      \save@sf@q{\set@low@box{\textquoteright/}}%
1996      \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.

```

1997 \ProvideTextCommandDefault{\quotesinglbase}{%
1998      \UseTextSymbol{OT1}{\quotesinglbase}}

```

**\guillemetleft**

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

```

1999 \ProvideTextCommand{\guillemetleft}{OT1}{%
2000      \ifmmode
2001          \ll
2002      \else
2003          \save@sf@q{\nobreak
2004              \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2005          \fi}
2006 \ProvideTextCommand{\guillemetright}{OT1}{%
2007      \ifmmode
2008          \gg
2009      \else
2010          \save@sf@q{\nobreak
2011              \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%

```

```

2012 \fi}
2013 \ProvideTextCommand{\guillemotleft}{OT1}{%
2014 \ifmmode
2015 \ll
2016 \else
2017 \save@sf@q{\nobreak
2018 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2019 \fi}
2020 \ProvideTextCommand{\guillemotright}{OT1}{%
2021 \ifmmode
2022 \gg
2023 \else
2024 \save@sf@q{\nobreak
2025 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2026 \fi}

```

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

```

2027 \ProvideTextCommandDefault{\guillemetleft}{%
2028 \UseTextSymbol{OT1}{\guillemetleft}}
2029 \ProvideTextCommandDefault{\guillemetright}{%
2030 \UseTextSymbol{OT1}{\guillemetright}}
2031 \ProvideTextCommandDefault{\guillemotleft}{%
2032 \UseTextSymbol{OT1}{\guillemotleft}}
2033 \ProvideTextCommandDefault{\guillemotright}{%
2034 \UseTextSymbol{OT1}{\guillemotright}}

```

#### **\guilsinglleft**

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

```

2035 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2036 \ifmmode
2037 <%
2038 \else
2039 \save@sf@q{\nobreak
2040 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2041 \fi}
2042 \ProvideTextCommand{\guilsinglright}{OT1}{%
2043 \ifmmode
2044 >%
2045 \else
2046 \save@sf@q{\nobreak
2047 \raise.2ex\hbox{$\scriptscriptstyle>$}\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{\guilsinglleft}{%
2050 \UseTextSymbol{OT1}{\guilsinglleft}}
2051 \ProvideTextCommandDefault{\guilsinglright}{%
2052 \UseTextSymbol{OT1}{\guilsinglright}}

```

## **4.15.2. Letters**

### **\ij**

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

```

2053 \DeclareTextCommand{\ij}{OT1}{%
2054 i\kern-0.02em\bbl@allowhyphens j}
2055 \DeclareTextCommand{\IJ}{OT1}{%
2056 I\kern-0.02em\bbl@allowhyphens J}
2057 \DeclareTextCommand{\ij}{T1}{\char188}
2058 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2059 \ProvideTextCommandDefault{\ij}{%
2060   \UseTextSymbol{OT1}{\ij}}
2061 \ProvideTextCommandDefault{\IJ}{%
2062   \UseTextSymbol{OT1}{\IJ}}
```

**\dj**

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

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

```
2063 \def\crrtic@{\hrule height0.1ex width0.3em}
2064 \def\crttic@{\hrule height0.1ex width0.33em}
2065 \def\ddj@{%
2066   \setbox0\hbox{d}\dimen@=\ht0
2067   \advance\dimen@lex
2068   \dimen@.45\dimen@
2069   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2070   \advance\dimen@ii.5ex
2071   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2072 \def\DDJ@{%
2073   \setbox0\hbox{D}\dimen@=.55\ht0
2074   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2075   \advance\dimen@ii.15ex % correction for the dash position
2076   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2077   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2078   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2079 %
2080 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2081 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2082 \ProvideTextCommandDefault{\dj}{%
2083   \UseTextSymbol{OT1}{\dj}}
2084 \ProvideTextCommandDefault{\DJ}{%
2085   \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.

```
2086 \DeclareTextCommand{\SS}{OT1}{SS}
2087 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 4.15.3. Shorthands for quotation marks

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

**\glq**

**\grq** The ‘german’ single quotes.

```
2088 \ProvideTextCommandDefault{\glq}{%
2089   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2090 \ProvideTextCommand{\grq}{T1}{%
2091   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2092 \ProvideTextCommand{\grq}{TU}{%
2093   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2094 \ProvideTextCommand{\grq}{OT1}{%
2095   \save@sf@q{\kern-.0125em
2096     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2097   }
```

```

2097 \kern.07em\relax}}
2098 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

#### **\glqq**

**\grqq** The ‘german’ double quotes.

```

2099 \ProvideTextCommandDefault{\glqq}{%
2100 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2101 \ProvideTextCommand{\grqq}{T1}{%
2102 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2103 \ProvideTextCommand{\grqq}{TU}{%
2104 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2105 \ProvideTextCommand{\grqq}{0T1}{%
2106 \save@sf@q{\kern-.07em
2107 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2108 \kern.07em\relax}}
2109 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

#### **\flq**

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

```

2110 \ProvideTextCommandDefault{\flq}{%
2111 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2112 \ProvideTextCommandDefault{\frq}{%
2113 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

#### **\flqq**

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

```

2114 \ProvideTextCommandDefault{\flqq}{%
2115 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2116 \ProvideTextCommandDefault{\frqq}{%
2117 \textormath{\guillemetright}{\mbox{\guillemetright}}}

```

### 4.15.4. Umlauts and tremas

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

#### **\umlauthigh**

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

```

2118 \def\umlauthigh{%
2119 \def\bbl@umlauta##1{\leavevmode\bgroup%
2120 \accent\csname\f@encoding dqpos\endcsname
2121 ##1\bbl@allowhyphens\egroup}%
2122 \let\bbl@umlaute\bbl@umlauta}
2123 \def\umlautlow{%
2124 \def\bbl@umlauta{\protect\lower@umlaut}}
2125 \def\umlautelow{%
2126 \def\bbl@umlaute{\protect\lower@umlaut}}
2127 \umlauthigh

```

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

```
2128 \expandafter\ifx\csname U@D\endcsname\relax
2129 \csname newdimen\endcsname\U@D
2130 \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.

```
2131 \def\lower@umlaut#1{%
2132 \leavevmode\bgroup
2133 \U@D lex%
2134 {\setbox\z@\hbox{%
2135 \char\csname f@encoding dqpos\endcsname}%
2136 \dimen@ -.45ex\advance\dimen@ \ht\z@
2137 \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2138 \accent\csname f@encoding dqpos\endcsname
2139 \fontdimen5\font\U@D #1%
2140 \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).

```
2141 \AtBeginDocument{%
2142 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2143 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2144 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2145 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2146 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2147 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2148 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2149 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2150 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2151 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2152 \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.

```
2153 \ifx\l@english\@undefined
2154 \chardef\l@english\z@
2155 \fi
2156 % The following is used to cancel rules in ini files (see Amharic).
2157 \ifx\l@unhyphenated\@undefined
2158 \newlanguage\l@unhyphenated
2159 \fi
```

## 4.16. Layout

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

```
2160 \bbl@trace{Bidi layout}
2161 \providecommand\IfBabelLayout[3]{#3}%
```

## 4.17. Load engine specific macros

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

```
2162 \bbl@trace{Input engine specific macros}
2163 \ifcase\bbl@engine
2164   \input txtbabel.def
2165 \or
2166   \input luababel.def
2167 \or
2168   \input xebabel.def
2169 \fi
2170 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2171 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2172 \ifx\babelposthyphenation\undefined
2173   \let\babelposthyphenation\babelprehyphenation
2174   \let\babelpatterns\babelprehyphenation
2175   \let\babelcharproperty\babelprehyphenation
2176 \fi
2177 </package | core>
```

## 4.18. Creating and modifying languages

Continue with  $\LaTeX$  only.

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

```
2178 < *package>
2179 \bbl@trace{Creating languages and reading ini files}
2180 \let\bbl@extend@ini@gobble
2181 \newcommand\babelprovide[2][]{%
2182   \let\bbl@savelangname\languagename
2183   \edef\bbl@savelocaleid{\the\localeid}%
2184   % Set name and locale id
2185   \edef\languagename{#2}%
2186   \bbl@id@assign
2187   % Initialize keys
2188   \bbl@vforeach{captions,date,import,main,script,language,%
2189     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2190     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2191     Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2192     {\bbl@csarg\let{KVP@##1}\@nnil}%
2193   \global\let\bbl@release@transforms@empty
2194   \global\let\bbl@release@casing@empty
2195   \let\bbl@calendars@empty
2196   \global\let\bbl@inidata@empty
2197   \global\let\bbl@extend@ini@gobble
2198   \global\let\bbl@included@inis@empty
2199   \gdef\bbl@key@list{;}%
2200   \bbl@ifunset{\bbl@passto@#2}%
2201     {\def\bbl@tempa{#1}}%
2202     {\bbl@exp{\def\\bbl@tempa{[\bbl@passto@#2],\unexpanded{#1}}}}%
2203   \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2204     \in@{/}{##1}% With /, (re)sets a value in the ini
2205     \ifin@
2206       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2207       \bbl@renewinikey##1\@{##2}%
2208     \else
2209       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2210         \bbl@error{unknown-provide-key}{##1}{}{}%
2211       \fi
2212       \bbl@csarg\def{KVP@##1}{##2}%
2213     \fi}%
2213 \fi%
```

```

2214 \chardef\bbbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2215 \bbbl@ifunset{date#2}\z@{\bbbl@ifunset{bbbl@llevel@#2}\@ne\tw@}%
2216 % == init ==
2217 \ifx\bbbl@screset\@undefined
2218 \bbbl@ldfinit
2219 \fi
2220 % ==
2221 \ifx\bbbl@KVP@import\@nnil\else \ifx\bbbl@KVP@import\@nnil
2222 \def\bbbl@KVP@import{\@empty}%
2223 \fi\fi
2224 % == date (as option) ==
2225 % \ifx\bbbl@KVP@date\@nnil\else
2226 % \fi
2227 % ==
2228 \let\bbbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2229 \ifcase\bbbl@howloaded
2230 \let\bbbl@lbfkflag\@empty % new
2231 \else
2232 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2233 \let\bbbl@lbfkflag\@empty
2234 \fi
2235 \ifx\bbbl@KVP@import\@nnil\else
2236 \let\bbbl@lbfkflag\@empty
2237 \fi
2238 \fi
2239 % == import, captions ==
2240 \ifx\bbbl@KVP@import\@nnil\else
2241 \bbbl@exp{\bbbl@ifblank{\bbbl@KVP@import}}%
2242 {\ifx\bbbl@initload\relax
2243 \begingroup
2244 \def\BabelBeforeIni##1##2{\gdef\bbbl@KVP@import{##1}\endinput}%
2245 \bbbl@input@texini{##2}%
2246 \endgroup
2247 \else
2248 \xdef\bbbl@KVP@import{\bbbl@initload}%
2249 \fi}%
2250 {}%
2251 \let\bbbl@KVP@date\@empty
2252 \fi
2253 \let\bbbl@KVP@captions@\bbbl@KVP@captions
2254 \ifx\bbbl@KVP@captions\@nnil
2255 \let\bbbl@KVP@captions\bbbl@KVP@import
2256 \fi
2257 % ==
2258 \ifx\bbbl@KVP@transforms\@nnil\else
2259 \bbbl@replace\bbbl@KVP@transforms{ }{,}%
2260 \fi
2261 % == Load ini ==
2262 \ifcase\bbbl@howloaded
2263 \bbbl@provide@new{##2}%
2264 \else
2265 \bbbl@ifblank{##1}%
2266 {}% With \bbbl@load@basic below
2267 {\bbbl@provide@renew{##2}}%
2268 \fi
2269 % Post tasks
2270 % -----
2271 % == subsequent calls after the first provide for a locale ==
2272 \ifx\bbbl@inidata\@empty\else
2273 \bbbl@extend@ini{##2}%
2274 \fi
2275 % == ensure captions ==
2276 \ifx\bbbl@KVP@captions\@nnil\else

```



```

2277 \bbl@ifunset{bbl@extracaps@#2}%
2278 {\bbl@exp{\bbl@babelensure[exclude=\today]{#2}}}%
2279 {\bbl@exp{\bbl@babelensure[exclude=\today,
2280 include=\bbl@extracaps@#2]}{#2}}%
2281 \bbl@ifunset{bbl@ensure@language}%
2282 {\bbl@exp{%
2283 \\\DeclareRobustCommand\<bbl@ensure@language>[1]{%
2284 \\\foreignlanguage{language}%
2285 {###1}}}%
2286 }%
2287 \bbl@exp{%
2288 \\\bbl@tglobal\<bbl@ensure@language>%
2289 \\\bbl@tglobal\<bbl@ensure@language\space>%
2290 \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.

```

2291 \bbl@load@basic{#2}%
2292 % == script, language ==
2293 % Override the values from ini or defines them
2294 \ifx\bbl@KVP@script\@nnil\else
2295 \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2296 \fi
2297 \ifx\bbl@KVP@language\@nnil\else
2298 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2299 \fi
2300 \ifcase\bbl@engine\or
2301 \bbl@ifunset{bbl@chrng@language}{}%
2302 {\directlua{
2303 Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2304 \fi
2305 % == Line breaking: intraspace, intrapenalty ==
2306 % For CJK, East Asian, Southeast Asian, if interspace in ini
2307 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2308 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2309 \fi
2310 \bbl@provide@intraspace
2311 % == Line breaking: justification ==
2312 \ifx\bbl@KVP@justification\@nnil\else
2313 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2314 \fi
2315 \ifx\bbl@KVP@linebreaking\@nnil\else
2316 \bbl@xin@{\bbl@KVP@linebreaking,%
2317 {,elongated,kashida,cjk,padding,unhyphenated},}%
2318 \ifin@
2319 \bbl@csarg\xdef
2320 {lnbrk@language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2321 \fi
2322 \fi
2323 \bbl@xin@{/e}{\bbl@cl{lnbrk}}%
2324 \ifin@\else\bbl@xin@{/k}{\bbl@cl{lnbrk}}\fi
2325 \ifin@\bbl@arabicjust\fi
2326 \bbl@xin@{/p}{\bbl@cl{lnbrk}}%
2327 \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2328 % == Line breaking: hyphenate.other.(locale|script) ==
2329 \ifx\bbl@lbfkflag\@empty
2330 \bbl@ifunset{bbl@hyotl@language}{}%
2331 {\bbl@csarg\bbl@replace{hyotl@language}{ }{ },}%
2332 \bbl@startcommands*{language}{}%
2333 \bbl@csarg\bbl@foreach{hyotl@language}{%
2334 \ifcase\bbl@engine
2335 \ifnum##1<257

```

```

2336         \SetHyphenMap{\BabelLower{##1}{##1}}%
2337     \fi
2338     \else
2339         \SetHyphenMap{\BabelLower{##1}{##1}}%
2340     \fi}%
2341 \bbl@endcommands}%
2342 \bbl@ifunset{\bbl@hyots@language}{%
2343     {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2344     \bbl@csarg\bbl@foreach{hyots@language}{%
2345         \ifcase\bbl@engine
2346         \ifnum##1<257
2347             \global\lccode##1=##1\relax
2348         \fi
2349         \else
2350             \global\lccode##1=##1\relax
2351         \fi}}}%
2352 \fi
2353 % == Counters: maparabic ==
2354 % Native digits, if provided in ini (TeX level, xe and lua)
2355 \ifcase\bbl@engine\else
2356     \bbl@ifunset{\bbl@dgnat@language}{%
2357         {\expandafter\ifx\csname\bbl@dgnat@language\endcsname\@empty\else
2358             \expandafter\expandafter\expandafter
2359             \bbl@setdigits\csname\bbl@dgnat@language\endcsname
2360             \ifx\bbl@KVP@maparabic\@nnil\else
2361                 \ifx\bbl@latinarabic\@undefined
2362                     \expandafter\let\expandafter\@arabic
2363                     \csname\bbl@counter@language\endcsname
2364                 \else % i.e., if layout=counters, which redefines \@arabic
2365                     \expandafter\let\expandafter\bbl@latinarabic
2366                     \csname\bbl@counter@language\endcsname
2367                 \fi
2368             \fi
2369         \fi}%
2370 \fi
2371 % == Counters: mapdigits ==
2372 % > luababel.def
2373 % == Counters: alph, Alph ==
2374 \ifx\bbl@KVP@alph\@nnil\else
2375     \bbl@exp{%
2376         \\bbl@add<\bbl@preextras@language>{%
2377             \\babel@save\\@alph
2378             \let\\@alph<\bbl@cntr@bbl@KVP@alph @language>}}%
2379 \fi
2380 \ifx\bbl@KVP@Alph\@nnil\else
2381     \bbl@exp{%
2382         \\bbl@add<\bbl@preextras@language>{%
2383             \\babel@save\\@Alph
2384             \let\\@Alph<\bbl@cntr@bbl@KVP@Alph @language>}}%
2385 \fi
2386 % == Casing ==
2387 \bbl@release@casing
2388 \ifx\bbl@KVP@casing\@nnil\else
2389     \bbl@csarg\xdef{casing@language}%
2390     {\@nameuse{\bbl@casing@language}\bbl@maybextx\bbl@KVP@casing}%
2391 \fi
2392 % == Calendars ==
2393 \ifx\bbl@KVP@calendar\@nnil
2394     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2395 \fi
2396 \def\bbl@tempe##1 ##2\@{ % Get first calendar
2397     \def\bbl@tempa{##1}}%
2398     \bbl@exp{\\bbl@tempe\bbl@KVP@calendar\space\\@}%

```

```

2399 \def\bbl@tempe##1.##2.##3\@{%
2400   \def\bbl@tempc{##1}%
2401   \def\bbl@tempb{##2}}%
2402 \expandafter\bbl@tempe\bbl@tempa..\@@
2403 \bbl@csarg\edef\calpr@\language\name}%
2404 \ifx\bbl@tempc\@empty\else
2405   calendar=\bbl@tempc
2406 \fi
2407 \ifx\bbl@tempb\@empty\else
2408   ,variant=\bbl@tempb
2409 \fi}%
2410 % == engine specific extensions ==
2411 % Defined in XXXbabel.def
2412 \bbl@provide@extra{#2}%
2413 % == require.babel in ini ==
2414 % To load or reload the babel-*.tex, if require.babel in ini
2415 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2416   \bbl@ifunset{\bbl@rqtex@\language\name}}{%
2417     {\expandafter\ifx\csname\bbl@rqtex@\language\name\endcsname\@empty\else
2418       \let\BabelBeforeIni\@gobbletwo
2419       \chardef\atcatcode=\catcode\@
2420       \catcode\@=11\relax
2421       \def\CurrentOption{#2}%
2422       \bbl@input@texini{\bbl@cs{rqtex@\language\name}}%
2423       \catcode\@=\atcatcode
2424       \let\atcatcode\relax
2425       \global\bbl@csarg\let{rqtex@\language\name}\relax
2426     \fi}%
2427 \bbl@foreach\bbl@calendars{%
2428   \bbl@ifunset{\bbl@ca-##1}{%
2429     \chardef\atcatcode=\catcode\@
2430     \catcode\@=11\relax
2431     \InputIfFileExists{babel-ca-##1.tex}{\fi}%
2432     \catcode\@=\atcatcode
2433     \let\atcatcode\relax}%
2434   }}%
2435 \fi
2436 % == frenchspacing ==
2437 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2438 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2439 \ifin@
2440   \bbl@extras@wrap{\bbl@pre@fs}%
2441   {\bbl@pre@fs}%
2442   {\bbl@post@fs}%
2443 \fi
2444 % == transforms ==
2445 % > luababel.def
2446 \def\CurrentOption{#2}%
2447 \@nameuse{\bbl@icsave{#2}}%
2448 % == main ==
2449 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2450   \let\language\bbl@savelangname
2451   \chardef\localeid\bbl@savelocaleid\relax
2452 \fi
2453 % == hyphenrules (apply if current) ==
2454 \ifx\bbl@KVP@hyphenrules\@nnil\else
2455   \ifnum\bbl@savelocaleid=\localeid
2456     \language\@nameuse{l@\language\name}%
2457   \fi
2458 \fi}

```

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

```

2459 \def\bbl@provide@new#1{%
2460 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2461 \namedef{extras#1}{}%
2462 \namedef{noextras#1}{}%
2463 \bbl@startcommands*{#1}{captions}%
2464 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2465 \def\bbl@tempb##1{% elt for \bbl@captionslist
2466 \ifx##1\@nnil\else
2467 \bbl@exp{%
2468 \\\SetString\\##1{%
2469 \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2470 \expandafter\bbl@tempb
2471 \fi}%
2472 \expandafter\bbl@tempb\bbl@captionslist\@nnil
2473 \else
2474 \ifx\bbl@initoload\relax
2475 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2476 \else
2477 \bbl@read@ini{\bbl@initoload}2% % Same
2478 \fi
2479 \fi
2480 \StartBabelCommands*{#1}{date}%
2481 \ifx\bbl@KVP@date\@nnil
2482 \bbl@exp{%
2483 \\\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2484 \else
2485 \bbl@savetoday
2486 \bbl@savestate
2487 \fi
2488 \bbl@endcommands
2489 \bbl@load@basic{#1}%
2490 % == hyphenmins == (only if new)
2491 \bbl@exp{%
2492 \gdef<#1hyphenmins>{%
2493 {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2494 {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2495 % == hyphenrules (also in renew) ==
2496 \bbl@provide@hyphens{#1}%
2497 \ifx\bbl@KVP@main\@nnil\else
2498 \expandafter\main@language\expandafter{#1}%
2499 \fi}
2500 %
2501 \def\bbl@provide@renew#1{%
2502 \ifx\bbl@KVP@captions\@nnil\else
2503 \StartBabelCommands*{#1}{captions}%
2504 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2505 \EndBabelCommands
2506 \fi
2507 \ifx\bbl@KVP@date\@nnil\else
2508 \StartBabelCommands*{#1}{date}%
2509 \bbl@savetoday
2510 \bbl@savestate
2511 \EndBabelCommands
2512 \fi
2513 % == hyphenrules (also in new) ==
2514 \ifx\bbl@lbkflag\@empty
2515 \bbl@provide@hyphens{#1}%
2516 \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.

```

2517 \def\bbl@load@basic#1{%

```

```

2518 \ifcase\bb@howloaded\or\or
2519 \ifcase\csname bbl@llevel@\language\endcsname
2520 \bbl@csarg\let{lname@\language}\relax
2521 \fi
2522 \fi
2523 \bbl@ifunset{bbl@lname@#1}%
2524 {\def\BabelBeforeIni##1##2{%
2525 \begingroup
2526 \let\bbl@ini@captions@aux\@gobbletwo
2527 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2528 \bbl@read@ini{##1}l%
2529 \ifx\bbl@initoload\relax\endinput\fi
2530 \endgroup}%
2531 \begingroup % boxed, to avoid extra spaces:
2532 \ifx\bbl@initoload\relax
2533 \bbl@input@texini{#1}%
2534 \else
2535 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2536 \fi
2537 \endgroup}%
2538 {}

```

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

```

2539 \def\bbl@load@info#1{%
2540 \def\BabelBeforeIni##1##2{%
2541 \begingroup
2542 \bbl@read@ini{##1}0%
2543 \endinput % babel- .tex may contain onlypreamble's
2544 \endgroup}% % boxed, to avoid extra spaces:
2545 {\bbl@input@texini{#1}}

```

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.

```

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

```

```

2574 \fi
2575 \fi
2576 \bbl@ifunset{l@#1}%
2577 {\ifnum\@tempcnta=\m@ne
2578 \bbl@carg\adddialect{l@#1}\language
2579 \else
2580 \bbl@carg\adddialect{l@#1}\@tempcnta
2581 \fi}%
2582 {\ifnum\@tempcnta=\m@ne\else
2583 \global\bbl@carg\chardef{l@#1}\@tempcnta
2584 \fi}}

```

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

```

2585 \def\bbl@input@texini#1{%
2586 \bbl@bsphack
2587 \bbl@exp{%
2588 \catcode`\\%=14 \catcode`\\=\=0
2589 \catcode`\\{=1 \catcode`\\}=2
2590 \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2591 \catcode`\\%=the\catcode`\%relax
2592 \catcode`\\{=the\catcode`\{relax
2593 \catcode`\\}=the\catcode`\}relax
2594 \catcode`\\}=the\catcode`\}relax}%
2595 \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.

```

2596 \def\bbl@iniline#1\bbl@iniline{%
2597 \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@% ]
2598 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2599 \def\bbl@iniskip#1\@@{% if starts with ;
2600 \def\bbl@inistore#1=#2\@@% full (default)
2601 \bbl@trim@def\bbl@tempa{#1}%
2602 \bbl@trim\toks{#2}%
2603 \bbl@ifsamestring{\bbl@tempa}{\include}%
2604 {\bbl@read@subini{\the\toks}}%
2605 {\bbl@xin@{\bbl@section/\bbl@tempa};{\bbl@key@list}%
2606 \ifin@
2607 \bbl@xin@{,identification/include.}%
2608 {,\bbl@section/\bbl@tempa}%
2609 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2610 \bbl@exp{%
2611 \\g@addto@macro\\bbl@inidata{%
2612 \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2613 \fi}}
2614 \def\bbl@inistore@min#1=#2\@@% minimal (maybe set in \bbl@read@ini)
2615 \bbl@trim@def\bbl@tempa{#1}%
2616 \bbl@trim\toks{#2}%
2617 \bbl@xin@{.identification.}{.\bbl@section.}%
2618 \ifin@
2619 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2620 \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2621 \fi}

```

## 4.19. Main loop in 'provide'

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

minimal data for fonts; with \babel provide it's either 1 (without import) or 2 (which import). The value -1 is used with \DocumentMetadata.

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

When the language is being set based on the document metadata (#2 in \bbl@read@ini is -1), there is an interlude to get the name, after the data have been collected, and before it's processed.

```

2622 \def\bbl@loop@ini#1{%
2623   \loop
2624     \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2625     \endlinechar\m@ne
2626     \read#1 to \bbl@line
2627     \endlinechar\^^M
2628     \ifx\bbl@line\empty\else
2629       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2630     \fi
2631   \repeat}
2632 %
2633 \def\bbl@read@subini#1{%
2634   \ifx\bbl@readsubstream\undefined
2635     \csname newread\endcsname\bbl@readsubstream
2636   \fi
2637   \openin\bbl@readsubstream=babel-#1.ini
2638   \ifeof\bbl@readsubstream
2639     \bbl@error{no-ini-file}{#1}{}}%
2640   \else
2641     {\bbl@loop@ini\bbl@readsubstream}%
2642   \fi
2643   \closein\bbl@readsubstream}
2644 %
2645 \ifx\bbl@readstream\undefined
2646   \csname newread\endcsname\bbl@readstream
2647 \fi
2648 \def\bbl@read@ini#1#2{%
2649   \global\let\bbl@extend@ini@gobble
2650   \openin\bbl@readstream=babel-#1.ini
2651   \ifeof\bbl@readstream
2652     \bbl@error{no-ini-file}{#1}{}}%
2653   \else
2654     % == Store ini data in \bbl@inidata ==
2655     \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
2656     \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2657     \ifnum#2=\m@ne % Just for the info
2658       \edef\language{tag \bbl@metalang}%
2659     \fi
2660     \bbl@info{Importing
2661               \ifcase#2font and identification \or basic \fi
2662               data for \language\%
2663               from babel-#1.ini. Reported}%
2664     \ifnum#2<\@ne
2665       \global\let\bbl@inidata\empty
2666       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2667     \fi
2668     \def\bbl@section{identification}%
2669     \bbl@exp{%
2670       \\bbl@inistore tag.ini=#1\\@@
2671       \\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\@@}%
2672     \bbl@loop@ini\bbl@readstream
2673     % == Process stored data ==
2674     \ifnum#2=\m@ne
2675       \def\bbl@tempa##1 ##2\@{##1}% Get first name
2676       \def\bbl@elt##1##2##3{%
2677         \bbl@ifsamestring{identification/name.babel}{##1/##2}%

```

```

2678         {\edef\language\language{\bbl@tempa###3 \@}%
2679         \bbl@id@assign
2680         \def\bbl@elt###1###2###3{}}}%
2681         {}}}%
2682         \bbl@inidata
2683         \fi
2684         \bbl@csarg\xdef\l@ini{\language\language}{#1}%
2685         \bbl@read@ini@aux
2686         % == 'Export' data ==
2687         \bbl@ini@exports{#2}%
2688         \global\bbl@csarg\let\inidata@\language\language\bbl@inidata
2689         \global\let\bbl@inidata\@empty
2690         \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language\language}}%
2691         \bbl@to@global\bbl@ini@loaded
2692         \fi
2693         \closein\bbl@readstream}
2694 \def\bbl@read@ini@aux{%
2695   \let\bbl@savestrings\@empty
2696   \let\bbl@savetoday\@empty
2697   \let\bbl@savestate\@empty
2698   \def\bbl@elt###1###2###3{%
2699     \def\bbl@section{##1}%
2700     \in@{=date.}{=##1}% Find a better place
2701     \ifin@
2702       \bbl@ifunset\bbl@inikv{##1}%
2703       {\bbl@ini@calendar{##1}}}%
2704     {}}%
2705   \fi
2706   \bbl@ifunset\bbl@inikv{##1}{}%
2707   {\csname bbl@inikv##1\endcsname{##2}{##3}}}%
2708   \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.

```

2709 \def\bbl@extend@ini@aux#1{%
2710   \bbl@startcommands*{#1}{captions}%
2711   % Activate captions/... and modify exports
2712   \bbl@csarg\def\inikv@captions.licr{##1##2}%
2713   \setlocalecaption{#1}{##1}{##2}}%
2714   \def\bbl@inikv@captions##1##2{%
2715     \bbl@ini@captions@aux{##1}{##2}}%
2716   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2717   \def\bbl@exportkey##1##2##3{%
2718     \bbl@ifunset\bbl@kv{##2}{}%
2719     {\expandafter\ifx\csname bbl@kv{##2}\endcsname\@empty\else
2720       \bbl@exp{\global\let<bbl@##1\language>\<bbl@kv{##2}>}}%
2721     \fi}}%
2722   % As with \bbl@read@ini, but with some changes
2723   \bbl@read@ini@aux
2724   \bbl@ini@exports\tw@
2725   % Update inidata@lang by pretending the ini is read.
2726   \def\bbl@elt###1###2###3{%
2727     \def\bbl@section{##1}%
2728     \bbl@iniline##2=##3\bbl@iniline}%
2729     \csname bbl@inidata@#1\endcsname
2730     \global\bbl@csarg\let\inidata@#1\bbl@inidata
2731   \StartBabelCommands*{#1}{date}% And from the import stuff
2732   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2733   \bbl@savetoday
2734   \bbl@savestate
2735   \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2736 \def\bbl@ini@calendar#1{%

```



```

2737 \lowercase{\def\bbl@tempa{=#1=}}%
2738 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2739 \bbl@replace\bbl@tempa{=date.}{}%
2740 \in@{.licr=}{#1=}%
2741 \ifin@
2742 \ifcase\bbl@engine
2743 \bbl@replace\bbl@tempa{.licr=}{}%
2744 \else
2745 \let\bbl@tempa\relax
2746 \fi
2747 \fi
2748 \ifx\bbl@tempa\relax\else
2749 \bbl@replace\bbl@tempa{=}{}%
2750 \ifx\bbl@tempa\empty\else
2751 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2752 \fi
2753 \bbl@exp{%
2754 \def\<bbl@inikv@#1>####1####2{%
2755 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2756 \fi}

```

A key with a slash in `\babelprovide` replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in `\bbl@inistore` above).

```

2757 \def\bbl@renewinikv#1/#2\@@#3{%
2758 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2759 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2760 \bbl@trim\toks@{#3}% value
2761 \bbl@exp{%
2762 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2763 \\\g@addto@macro\\bbl@inidata{%
2764 \\\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

```

2765 \def\bbl@exportkey#1#2#3{%
2766 \bbl@ifunset{\bbl@kv@#2}%
2767 {\bbl@csarg\gdef{#1@\language\language}{#3}}%
2768 {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2769 \bbl@csarg\gdef{#1@\language\language}{#3}%
2770 \else
2771 \bbl@exp{\global\let\<bbl@#1@\language\language>\<bbl@kv@#2>}%
2772 \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

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

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

```

2773 \def\bbl@iniwarning#1{%
2774 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2775 {\bbl@warning{%
2776 From babel-\bbl@cs{lini@\language\language}.ini:\\%
2777 \bbl@cs{kv@identification.warning#1}\\%
2778 Reported }}}
2779 %

```

```

2780 \let\bbl@release@transforms\@empty
2781 \let\bbl@release@casing\@empty

```

Relevant keys are ‘exported’, i.e., global macros with short names are created with values taken from the corresponding keys. The number of exported keys depends on the loading level (#1): –1 and 0 only info (the identification section), 1 also basic (like linebreaking or character ranges), 2 also (re)new (with date and captions).

```

2782 \def\bbl@ini@exports#1{%
2783   % Identification always exported
2784   \bbl@iniwarning{}}%
2785   \ifcase\bbl@engine
2786     \bbl@iniwarning{.pdflatex}%
2787   \or
2788     \bbl@iniwarning{.lua\latex}%
2789   \or
2790     \bbl@iniwarning{.xel\latex}%
2791   \fi%
2792   \bbl@exportkey{lllevel}{identification.load.level}{}}%
2793   \bbl@exportkey{elname}{identification.name.english}{}}%
2794   \bbl@expf{\bbl@exportkey{lname}{identification.name.opentype}%
2795     {\csname bbl@elname@language\endcsname}}%
2796   \bbl@exportkey{tbc}{identification.tag.bcp47}{}}%
2797   \bbl@exportkey{casing}{identification.tag.bcp47}{}}%
2798   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}}%
2799   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2800   \bbl@exportkey{esname}{identification.script.name}{}}%
2801   \bbl@expf{\bbl@exportkey{sname}{identification.script.name.opentype}%
2802     {\csname bbl@esname@language\endcsname}}%
2803   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}}%
2804   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2805   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}}%
2806   \bbl@exportkey{vbc}{identification.variant.tag.bcp47}{}}%
2807   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}}%
2808   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}}%
2809   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}}%
2810   % Also maps bcp47 -> language\name
2811   \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language\name}%
2812   \ifcase\bbl@engine\or
2813     \directlua{%
2814       Babel.locale_props[\the\bbl@cs{id@language}].script
2815       = '\bbl@cl{sbc}}}%
2816   \fi
2817   % Conditional
2818   \ifnum#1>\z@      % -1 or 0 = only info, 1 = basic, 2 = (re)new
2819     \bbl@exportkey{calpr}{date.calendar.preferred}{}}%
2820     \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
2821     \bbl@exportkey{hyphr}{typography.hyphenrules}{}}%
2822     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2823     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2824     \bbl@exportkey{prehc}{typography.prehyphenchar}{}}%
2825     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}}%
2826     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}}%
2827     \bbl@exportkey{intsp}{typography.intraspace}{}}%
2828     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2829     \bbl@exportkey{chrng}{characters.ranges}{}}%
2830     \bbl@exportkey{quote}{characters.delimiters.quotes}{}}%
2831     \bbl@exportkey{dgnat}{numbers.digits.native}{}}%
2832     \ifnum#1=\tw@      % only (re)new
2833       \bbl@exportkey{rqtex}{identification.require.babel}{}}%
2834       \bbl@to\global\bbl@savetoday
2835       \bbl@to\global\bbl@savestate
2836       \bbl@savestrings
2837   \fi

```

```
2838 \fi}
```

## 4.20. Processing keys in ini

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

```
2839 \def\bbl@inikv#1#2{%      key=value
2840 \toks@{#2}%              This hides #'s from ini values
2841 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
2842 \let\bbl@inikv@identification\bbl@inikv
2843 \let\bbl@inikv@date\bbl@inikv
2844 \let\bbl@inikv@typography\bbl@inikv
2845 \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`.

```
2846 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2847 \def\bbl@inikv@characters#1#2{%
2848 \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2849 {\bbl@exp{%
2850 \\\g@addto@macro\\\bbl@release@casing{%
2851 \\\bbl@casemapping}{\languagename}{\unexpanded{#2}}}%
2852 {\in@{casing.}{$#1}% e.g., casing.Uv = uV
2853 \ifin@
2854 \lowercase{\def\bbl@tempb{#1}}%
2855 \bbl@replace\bbl@tempb{casing.}{}%
2856 \bbl@exp{\\\g@addto@macro\\\bbl@release@casing{%
2857 \\\bbl@casemapping
2858 {\\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}%
2859 \else
2860 \bbl@inikv{#1}{#2}%
2861 \fi}}
```

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

```
2862 \def\bbl@inikv@counters#1#2{%
2863 \bbl@ifsamestring{#1}{digits}%
2864 {\bbl@error{digits-is-reserved}{}}}%
2865 {}%
2866 \def\bbl@tempc{#1}%
2867 \bbl@trim@def{\bbl@tempb*}{#2}%
2868 \in@{.1$}{#1$}%
2869 \ifin@
2870 \bbl@replace\bbl@tempc{.1}{}%
2871 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
2872 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2873 \fi
2874 \in@{.F.}{#1}%
2875 \ifin@else\in@{.S.}{#1}\fi
2876 \ifin@
2877 \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
2878 \else
2879 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2880 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2881 \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
2882 \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.

```

2883 \ifcase\bbl@engine
2884 \bbl@csarg\def{inikv@captions.licr}#1#2{%
2885 \bbl@ini@captions@aux{#1}{#2}}
2886 \else
2887 \def\bbl@inikv@captions#1#2{%
2888 \bbl@ini@captions@aux{#1}{#2}}
2889 \fi

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

2890 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2891 \bbl@replace\bbl@tempa{.template}{}%
2892 \def\bbl@toreplace{#1}{}%
2893 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2894 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
2895 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2896 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname{}}%
2897 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
2898 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2899 \ifin@
2900 \@nameuse{bbl@patch\bbl@tempa}%
2901 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2902 \fi
2903 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2904 \ifin@
2905 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2906 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2907 \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\language}%
2908 {\fnum@\bbl@tempa}}%
2909 {\\\@nameuse{bbl@\bbl@tempa fmt@\\language}}}%
2910 \fi}
2911 %
2912 \def\bbl@ini@captions@aux#1#2{%
2913 \bbl@trim\def\bbl@tempa{#1}%
2914 \bbl@xin@{.template}{\bbl@tempa}%
2915 \ifin@
2916 \bbl@ini@captions@template{#2}\language
2917 \else
2918 \bbl@ifblank{#2}%
2919 {\bbl@exp{%
2920 \toks@{\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2921 {\bbl@trim\toks@{#2}}}%
2922 \bbl@exp{%
2923 \\\bbl@add\\bbl@savestrings{%
2924 \\\SetString\<\bbl@tempa name>{\the\toks@}}%
2925 \toks@expandafter{\bbl@captionslist}%
2926 \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}%
2927 \ifin@else
2928 \bbl@exp{%
2929 \\\bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}%
2930 \\\bbl@tglobal\<\bbl@extracaps@\language>}%
2931 \fi
2932 \fi}

```

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

```

2933 \def\bbl@list@the{%
2934 part,chapter,section,subsection,subsubsection,paragraph,%
2935 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2936 table,page,footnote,mpfootnote,mpfn}
2937 %
2938 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2939 \bbl@ifunset{bbl@map@#1@\language}%
2940 {\@nameuse{#1}}%
2941 {\@nameuse{bbl@map@#1@\language}}}
2942 %

```

```

2943 \def\bbl@inikv@labels#1#2{%
2944   \in@{.map}{#1}%
2945   \ifin@
2946     \ifx\bbl@KVP@labels\@nnil\else
2947       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
2948       \ifin@
2949         \def\bbl@tempc{#1}%
2950         \bbl@replace\bbl@tempc{.map}{}%
2951         \in@{, #2, }{, arabic, roman, Roman, alph, Alph, fnsymbol,}%
2952         \bbl@exp{%
2953           \gdef\<bbl@map@\bbl@tempc @\languagename>%
2954             {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
2955         \bbl@foreach\bbl@list@the{%
2956           \bbl@ifunset{the##1}{}%
2957           {\bbl@exp{\let\\bbl@tempd\<the##1>}%
2958             \bbl@exp{%
2959               \\bbl@sreplace\<the##1>%
2960                 {\<\bbl@tempc>{##1}}%
2961                 {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
2962               \\bbl@sreplace\<the##1>%
2963                 {\<\empty @\bbl@tempc>\<c@##1>}%
2964                 {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
2965               \\bbl@sreplace\<the##1>%
2966                 {\\\csname @\bbl@tempc\\endcsname\<c@##1>}%
2967                 {\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2968             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2969               \bbl@exp{\gdef\<the##1>{\{the##1\}}}%
2970             \fi}}%
2971       \fi
2972     \fi
2973 %
2974 \else
2975   % The following code is still under study. You can test it and make
2976   % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
2977   % language dependent.
2978   \in@{enumerate.}{#1}%
2979   \ifin@
2980     \def\bbl@tempa{#1}%
2981     \bbl@replace\bbl@tempa{enumerate.}{}%
2982     \def\bbl@toreplace{#2}%
2983     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
2984     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2985     \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
2986     \toks@{\expandafter\bbl@toreplace}%
2987     \bbl@exp{%
2988       \\bbl@add\<extras\languagename>{%
2989         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
2990         \def\<labelenum\romannumeral\bbl@tempa>\{the\toks@}\}%
2991       \\bbl@tglobal\<extras\languagename>}%
2992     \fi
2993   \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.

```

2994 \def\bbl@chapttype{chapter}
2995 \ifx\@makechapterhead\undefined
2996   \let\bbl@patchchapter\relax
2997 \else\ifx\thechapter\undefined
2998   \let\bbl@patchchapter\relax
2999 \else\ifx\ps@headings\undefined
3000   \let\bbl@patchchapter\relax

```

```

3001 \else
3002   \def\bbl@patchchapter{%
3003     \global\let\bbl@patchchapter\relax
3004     \gdef\bbl@chfmt{%
3005       \bbl@ifunset\bbl@bbl@chapttype fmt@\languagename}%
3006       {\@chapapp\space\thechapter}%
3007       {\@nameuse\bbl@bbl@chapttype fmt@\languagename}}}%
3008   \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3009   \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3010   \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3011   \bbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3012   \bbl@tglobal\appendix
3013   \bbl@tglobal\ps@headings
3014   \bbl@tglobal\chaptermark
3015   \bbl@tglobal\makechapterhead}
3016   \let\bbl@patchappendix\bbl@patchchapter
3017 \fi\fi\fi
3018 \ifx\@part\undefined
3019   \let\bbl@patchpart\relax
3020 \else
3021   \def\bbl@patchpart{%
3022     \global\let\bbl@patchpart\relax
3023     \gdef\bbl@partformat{%
3024       \bbl@ifunset\bbl@partfmt@\languagename}%
3025       {\partname\nobreakspace\thepart}%
3026       {\@nameuse\bbl@partfmt@\languagename}}}%
3027   \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3028   \bbl@tglobal\@part}
3029 \fi

```

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

```

3030 \let\bbl@calendar\@empty
3031 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3032 \def\bbl@localedate#1#2#3#4{%
3033   \begingroup
3034     \edef\bbl@they{#2}%
3035     \edef\bbl@them{#3}%
3036     \edef\bbl@thed{#4}%
3037     \edef\bbl@tempe{%
3038       \bbl@ifunset\bbl@calpr@\languagename}{\bbl@cl{calpr}},%
3039       #1}%
3040     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3041     \bbl@replace\bbl@tempe{ }{}%
3042     \bbl@replace\bbl@tempe{convert}{convert=}%
3043     \let\bbl@ld@calendar\@empty
3044     \let\bbl@ld@variant\@empty
3045     \let\bbl@ld@convert\relax
3046     \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld###1}{##2}}%
3047     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3048     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3049     \ifx\bbl@ld@calendar\@empty\else
3050       \ifx\bbl@ld@convert\relax\else
3051         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3052         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3053       \fi
3054     \fi
3055     \@nameuse\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3056     \edef\bbl@calendar{% Used in \month..., too
3057       \bbl@ld@calendar
3058       \ifx\bbl@ld@variant\@empty\else
3059         .\bbl@ld@variant
3060       \fi}%

```

```

3061 \bbl@cased
3062 {\@nameuse{bbl@date@\language name @\bbl@calendar}%
3063 \bbl@they\bbl@them\bbl@thed}%
3064 \endgroup}
3065 %
3066 \def\bbl@printdate#1{%
3067 \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}]
3068 \def\bbl@printdate@i#1[#2]#3#4#5{%
3069 \bbl@usedategroupttrue
3070 \@nameuse{bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}}
3071 %
3072 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3073 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3074 \bbl@trim@def\bbl@tempa{#1.#2}%
3075 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3076 {\bbl@trim@def\bbl@tempa{#3}%
3077 \bbl@trim\toks@{#5}%
3078 \@temptokena\expandafter{\bbl@savedate}%
3079 \bbl@exp{% Reverse order - in ini last wins
3080 \def\\bbl@savedate{%
3081 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3082 \the\@temptokena}}}%
3083 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3084 {\lowercase{\def\bbl@tempb{#6}}}%
3085 \bbl@trim@def\bbl@toreplace{#5}%
3086 \bbl@TG@@date
3087 \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3088 \ifx\bbl@savetoday@empty
3089 \bbl@exp{%
3090 \\AfterBabelCommands{%
3091 \gdef\<\language name date>{\\protect\<\language name date >}%
3092 \gdef\<\language name date >{\\bbl@printdate{\language name}}}%
3093 \def\\bbl@savetoday{%
3094 \\SetString\\today{%
3095 \<\language name date>[convert]%
3096 {\the\year}{\the\month}{\the\day}}}%
3097 \fi}%
3098 {}}}}

```

**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).

```

3099 \let\bbl@calendar@empty
3100 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3101 \@nameuse{bbl@ca@#2}#1\@@}
3102 \newcommand\babelDateSpace{\nobreakspace}
3103 \newcommand\babelDateDot{.\@@}
3104 \newcommand\babelDated[1]{\number#1}
3105 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3106 \newcommand\babelDateM[1]{\number#1}
3107 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3108 \newcommand\babelDateMMMM[1]{%
3109 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3110 \newcommand\babelDatey[1]{\number#1}%
3111 \newcommand\babelDateyy[1]{%
3112 \ifnum#1<10 0\number#1 %
3113 \else\ifnum#1<100 \number#1 %
3114 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3115 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3116 \else
3117 \bbl@error{limit-two-digits}{}}}%

```

```

3118 \fi\fi\fi\fi}}
3119 \newcommand\BabelDateyyy[1]{\number#1}}
3120 \newcommand\BabelDateU[1]{\number#1}}%
3121 \def\bb@l@replace@finish@iii#1{%
3122 \bb@l@exp{\def\#1####1####2####3{\the\toks@}}
3123 \def\bb@l@TG@date{%
3124 \bb@l@replace\bb@l@toreplace{[ ]}{\BabelDateSpace{}}%
3125 \bb@l@replace\bb@l@toreplace{[.]}{\BabelDateDot{}}%
3126 \bb@l@replace\bb@l@toreplace{[d]}{\BabelDated{####3}}%
3127 \bb@l@replace\bb@l@toreplace{[dd]}{\BabelDatedd{####3}}%
3128 \bb@l@replace\bb@l@toreplace{[M]}{\BabelDateM{####2}}%
3129 \bb@l@replace\bb@l@toreplace{[MM]}{\BabelDateMM{####2}}%
3130 \bb@l@replace\bb@l@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3131 \bb@l@replace\bb@l@toreplace{[y]}{\BabelDatey{####1}}%
3132 \bb@l@replace\bb@l@toreplace{[yy]}{\BabelDateyy{####1}}%
3133 \bb@l@replace\bb@l@toreplace{[yyy]}{\BabelDateyyy{####1}}%
3134 \bb@l@replace\bb@l@toreplace{[U]}{\BabelDateU{####1}}%
3135 \bb@l@replace\bb@l@toreplace{[y]}{\bb@l@datecctr{####1}}%
3136 \bb@l@replace\bb@l@toreplace{[U]}{\bb@l@datecctr{####1}}%
3137 \bb@l@replace\bb@l@toreplace{[m]}{\bb@l@datecctr{####2}}%
3138 \bb@l@replace\bb@l@toreplace{[d]}{\bb@l@datecctr{####3}}%
3139 \bb@l@replace@finish@iii\bb@l@toreplace}
3140 \def\bb@l@datecctr{\expandafter\bb@l@xdatecctr\expandafter}
3141 \def\bb@l@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

## 4.21. French spacing (again)

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

```

3142 \AddToHook{begindocument/before}{%
3143 \let\bb@l@normalsf\normalsfcodes
3144 \let\normalsfcodes\relax}
3145 \AtBeginDocument{%
3146 \ifx\bb@l@normalsf\@empty
3147 \ifnum\sfcodes\@.=\@m
3148 \let\normalsfcodes\frenchspacing
3149 \else
3150 \let\normalsfcodes\nonfrenchspacing
3151 \fi
3152 \else
3153 \let\normalsfcodes\bb@l@normalsf
3154 \fi}

```

### Transforms.

Process the transforms read from ini files, converts them to a form close to the user interface (with `\babelprehyphenation` and `\babelposthyphenation`), wrapped with `\bb@l@transforms@aux` ...`\relax`, and stores them in `\bb@l@release@transforms`. However, since building a list enclosed in braces isn't trivial, the replacements are added after a comma, and then `\bb@l@transforms@aux` adds the braces.

```

3155 \bb@l@csarg\let{inikv@transforms.prehyphenation}\bb@l@inikv
3156 \bb@l@csarg\let{inikv@transforms.posthyphenation}\bb@l@inikv
3157 \def\bb@l@transforms@aux#1#2#3#4,#5\relax{%
3158 #1[#2]{#3}{#4}{#5}}
3159 \begingroup
3160 \catcode`\%=12
3161 \catcode`\&=14
3162 \gdef\bb@l@transforms#1#2#3{\&%
3163 \directlua{
3164 local str = [==[#2]==]
3165 str = str:gsub('%.%d+%.%d+$', '')
3166 token.set_macro('babeltempa', str)
3167 }&%
3168 \def\babeltempc{}}&%

```



```

3169 \bbl@xin@{\babeltempa,}{,\bbl@KVP@transforms,}&%
3170 \ifin@else
3171 \bbl@xin@{\babeltempa,}{,\bbl@KVP@transforms,}&%
3172 \fi
3173 \ifin@
3174 \bbl@foreach\bbl@KVP@transforms{&%
3175 \bbl@xin@{\babeltempa,}{,##1,}&%
3176 \ifin@ &% font:font:transform syntax
3177 \directlua{
3178 local t = {}
3179 for m in string.gmatch('##1'..' ':'', '(.)') do
3180 table.insert(t, m)
3181 end
3182 table.remove(t)
3183 token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3184 }&%
3185 \fi}&%
3186 \in@{.0$}{#2$}&%
3187 \ifin@
3188 \directlua{&% (\attribute) syntax
3189 local str = string.match([[ \bbl@KVP@transforms]],
3190 '%([^(%[]-)%[^(%[]-)]-\babeltempa')
3191 if str == nil then
3192 token.set_macro('babeltempb', '')
3193 else
3194 token.set_macro('babeltempb', ',attribute=' .. str)
3195 end
3196 }&%
3197 \toks@{#3}&%
3198 \bbl@exp{&%
3199 \\g@addto@macro\\bbl@release@transforms{&%
3200 \relax &% Closes previous \bbl@transforms@aux
3201 \\bbl@transforms@aux
3202 \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3203 {\language\the\toks@}}&%
3204 \else
3205 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3206 \fi
3207 \fi}
3208 \endgroup

```

## 4.22. Handle language system

The language system (i.e., Language and Script) to be used when defining a font or setting the direction are set with the following macros. It also deals with unhyphenated line breaking in xetex (e.g., Thai and traditional Sanskrit), which is done with a hack at the font level because this engine doesn't support it.

```

3209 \def\bbl@provide@lsys#1{%
3210 \bbl@ifunset\bbl@lname@#1{%
3211 {\bbl@load@info{#1}}%
3212 }%
3213 \bbl@csarg\let{lsys@#1}\@empty
3214 \bbl@ifunset\bbl@sname@#1{\bbl@csarg\gdef{sname@#1}{Default}}{%
3215 \bbl@ifunset\bbl@sotf@#1{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3216 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3217 \bbl@ifunset\bbl@lname@#1{%
3218 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3219 \ifcase\bbl@engine\or\or
3220 \bbl@ifunset\bbl@prehc@#1{%
3221 {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3222 }%
3223 {\ifx\bbl@xenohyph\undefined
3224 \global\let\bbl@xenohyph\bbl@xenohyph@

```



```

3274 \newcommand\locaenumerals[2]{\bbl@cs{cnt@#1@\language}\{#2}}
3275 \def\bbl@locaecnt#1#2{\locaenumerals{#2}{#1}}
3276 \newcommand\localecounter[2]{%
3277   \expandafter\bbl@locaecnt
3278   \expandafter{\number\csname c@#2\endcsname}\{#1}}
3279 \def\bbl@alphnumeral#1#2{%
3280   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3281 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3282   \ifcase\@car#8\@nil\or    % Currently <10000, but prepared for bigger
3283     \bbl@alphnumeral@ii{#9}000000#1\or
3284     \bbl@alphnumeral@ii{#9}000000#1#2\or
3285     \bbl@alphnumeral@ii{#9}000000#1#2#3\or
3286     \bbl@alphnumeral@ii{#9}000000#1#2#3#4\else
3287     \bbl@alphnum@invalid{>9999}%
3288   \fi}
3289 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3290   \bbl@ifunset{bbl@cnt@#1.F.\number#5#6#7#8@\language}%
3291     {\bbl@cs{cnt@#1.4@\language}\{#5}%
3292     \bbl@cs{cnt@#1.3@\language}\{#6}%
3293     \bbl@cs{cnt@#1.2@\language}\{#7}%
3294     \bbl@cs{cnt@#1.1@\language}\{#8}%
3295     \ifnum#6#7#8>\z@
3296       \bbl@ifunset{bbl@cnt@#1.S.321@\language}\{}}%
3297     {\bbl@cs{cnt@#1.S.321@\language}\{}}%
3298   \fi}%
3299   {\bbl@cs{cnt@#1.F.\number#5#6#7#8@\language}}}}
3300 \def\bbl@alphnum@invalid#1{%
3301   \bbl@error{alphabetic-too-large}{#1}\{}}

```

## 4.24. Casing

```

3302 \newcommand\BabelUppercaseMapping[3]{%
3303   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3304 \newcommand\BabelTitlecaseMapping[3]{%
3305   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3306 \newcommand\BabelLowercaseMapping[3]{%
3307   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

  The parser for casing and casing. (variant).
3308 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3309   \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3310 \else
3311   \def\bbl@utftocode#1{\expandafter\string#1}
3312 \fi
3313 \def\bbl@casemapping#1#2#3{% 1:variant
3314   \def\bbl@tempa##1 ##2{% Loop
3315     \bbl@casemapping@i{##1}%
3316     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3317   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3318   \def\bbl@tempe{0}% Mode (upper/lower...)
3319   \def\bbl@tempc{#3}% Casing list
3320   \expandafter\bbl@tempa\bbl@tempc\@empty}
3321 \def\bbl@casemapping@i#1{%
3322   \def\bbl@tempb{#1}%
3323   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3324     \@nameuse{regex_replace_all:nnN}%
3325     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\{\}\bbl@tempb
3326   \else
3327     \@nameuse{regex_replace_all:nnN}{.}{\{\}\bbl@tempb
3328   \fi
3329   \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3330 \def\bbl@casemapping@ii#1#2#3\@@{%
3331   \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3332   \ifin@

```

```

3333 \edef\bbbl@tempe{%
3334 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3335 \else
3336 \ifcase\bbbl@tempe\relax
3337 \DeclareUppercaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3338 \DeclareLowercaseMapping[\bbbl@templ]{\bbbl@uftocode{#2}}{#1}%
3339 \or
3340 \DeclareUppercaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3341 \or
3342 \DeclareLowercaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3343 \or
3344 \DeclareTitlecaseMapping[\bbbl@templ]{\bbbl@uftocode{#1}}{#2}%
3345 \fi
3346 \fi}

```

## 4.25. Getting info

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

```

3347 \def\bbbl@localeinfo#1#2{%
3348 \bbbl@ifunset{bbbl@info@#2}{#1}%
3349 {\bbbl@ifunset{bbbl@csname bbbl@info@#2\endcsname @\languagename}{#1}%
3350 {\bbbl@cs{\csname bbbl@info@#2\endcsname @\languagename}}}%
3351 \newcommand\localeinfo[1]{%
3352 \ifx*#1\@empty
3353 \bbbl@afterelse\bbbl@localeinfo{%
3354 \else
3355 \bbbl@localeinfo
3356 {\bbbl@error{no-ini-info}{}}{}}}%
3357 {#1}%
3358 \fi}
3359 % \@namedef{bbbl@info@name.locale}{lcname}
3360 \@namedef{bbbl@info@tag.ini}{lini}
3361 \@namedef{bbbl@info@name.english}{elname}
3362 \@namedef{bbbl@info@name.opentype}{lname}
3363 \@namedef{bbbl@info@tag.bcp47}{tbcpl}
3364 \@namedef{bbbl@info@language.tag.bcp47}{lbcpl}
3365 \@namedef{bbbl@info@tag.opentype}{lotf}
3366 \@namedef{bbbl@info@script.name}{esname}
3367 \@namedef{bbbl@info@script.name.opentype}{sname}
3368 \@namedef{bbbl@info@script.tag.bcp47}{sbcp}
3369 \@namedef{bbbl@info@script.tag.opentype}{sotf}
3370 \@namedef{bbbl@info@region.tag.bcp47}{rbcp}
3371 \@namedef{bbbl@info@variant.tag.bcp47}{vbcp}
3372 \@namedef{bbbl@info@extension.t.tag.bcp47}{extt}
3373 \@namedef{bbbl@info@extension.u.tag.bcp47}{extu}
3374 \@namedef{bbbl@info@extension.x.tag.bcp47}{extx}

```

With version 3.75 `\BabelEnsureInfo` is executed always, but there is an option to disable it. Since the info in ini files are always loaded, it has been made no-op in version 25.8.

```

3375 <<*More package options>> ≡
3376 \DeclareOption{ensureinfo=off}{}
3377 <</More package options>>
3378 \let\BabelEnsureInfo\relax

```

More general, but non-expandable, is `\getlocaleproperty`.

```

3379 \newcommand\getlocaleproperty{%
3380 \ifstar\bbbl@getproperty@*\bbbl@getproperty@x}
3381 \def\bbbl@getproperty@s#1#2#3{%
3382 \let#1\relax
3383 \def\bbbl@elt##1##2##3{%
3384 \bbbl@ifsamestring{##1/##2}{#3}%
3385 {\providecommand#1{##3}%
3386 \def\bbbl@elt####1####2####3{}}}%

```

```

3387     {}}%
3388     \bbl@cs{inidata@#2}}%
3389 \def\bbl@getproperty@x#1#2#3{%
3390     \bbl@getproperty@s{#1}{#2}{#3}%
3391     \ifx#1\relax
3392         \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3393     \fi}

```

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`.

```

3394 \let\bbl@ini@loaded\@empty
3395 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3396 \def\ShowLocaleProperties#1{%
3397     \typeout{}%
3398     \typeout{*** Properties for language '#1' ***}
3399     \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3400     \@nameuse{bbl@inidata@#1}%
3401     \typeout{*****}}

```

## 4.26. BCP 47 related commands

This macro is called by language selectors when the language isn't recognized. So, it's the core for (1) mapping from a BCP 27 tag to the actual language, if `bcp47.toname` is enabled (i.e., if `bbl@bcptoname` is true), and (2) lazy loading. With `autoload.bcp47` enabled *and* lazy loading, we must first build a name for the language, with the help of `autoload.bcp47.prefix`. Then we use `\provideprovide` passing the options set with `autoload.bcp47.options` (by default `import`). Finally, and if the locale has not been loaded before, we use `\provideprovide` with the language name as passed to the selector.

```

3402 \newif\ifbbl@bcpallowed
3403 \bbl@bcpallowedfalse
3404 \def\bbl@autoload@options{import}
3405 \def\bbl@provide@locale{%
3406     \ifx\babelprovide\@undefined
3407         \bbl@error{base-on-the-fly}{}{}%
3408     \fi
3409     \let\bbl@auxname\language
3410     \ifbbl@bcptoname
3411         \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
3412         {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
3413         \let\localename\language}%
3414     \fi
3415     \ifbbl@bcpallowed
3416         \expandafter\ifx\csname date\language\endcsname\relax
3417             \expandafter
3418             \bbl@bcplookup\language-\@empty-\@empty-\@empty@@
3419             \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3420                 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3421                 \let\localename\language
3422                 \expandafter\ifx\csname date\language\endcsname\relax
3423                     \let\bbl@initoload\bbl@bcp
3424                     \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3425                     \let\bbl@initoload\relax
3426                 \fi
3427                 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
3428             \fi
3429         \fi
3430     \fi
3431     \expandafter\ifx\csname date\language\endcsname\relax
3432         \IfFileExists{babel-\language.tex}%
3433         {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3434         {}%
3435     \fi}

```

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

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

```

3436 \providecommand\BCPdata{}
3437 \ifx\renewcommand\undefined\else
3438   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3439   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3440     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3441     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3442     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language\language}%
3443   \def\bbl@bcpdata@ii#1#2{%
3444     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3445     {\bbl@error{unknown-ini-field}{#1}{}}}%
3446     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3447     {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3448 \fi
3449 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3450 \@namedef{bbl@info@tag.tag.bcp47}{tbc} % For \BCPdata

```

## 5. Adjusting the Babel behavior

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

```

3451 \newcommand\babeladjust[1]{%
3452   \bbl@forkv{#1}{%
3453     \bbl@ifunset{bbl@ADJ@##1@##2}%
3454     {\bbl@cs{ADJ@##1}{##2}}%
3455     {\bbl@cs{ADJ@##1@##2}}}
3456 %
3457 \def\bbl@adjust@lua#1#2{%
3458   \ifvmode
3459     \ifnum\currentgrouplevel=\z@
3460       \directlua{ Babel.#2 }%
3461       \expandafter\expandafter\expandafter\@gobble
3462     \fi
3463   \fi
3464   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3465 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3466   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3467 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3468   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3469 \@namedef{bbl@ADJ@bidi.text@on}{%
3470   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3471 \@namedef{bbl@ADJ@bidi.text@off}{%
3472   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3473 \@namedef{bbl@ADJ@bidi.math@on}{%
3474   \let\bbl@noamsmath\@empty}
3475 \@namedef{bbl@ADJ@bidi.math@off}{%
3476   \let\bbl@noamsmath\relax}
3477 %
3478 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3479   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3480 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3481   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3482 %
3483 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3484   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3485 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3486   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3487 \@namedef{bbl@ADJ@linebreak.cjk@on}{%

```

```

3488 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3489 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3490 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3491 \@namedef{bbl@ADJ@justify.arabic@on}{%
3492 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3493 \@namedef{bbl@ADJ@justify.arabic@off}{%
3494 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3495 %
3496 \def\bbl@adjust@layout#1{%
3497 \ifvmode
3498 #1%
3499 \expandafter\@gobble
3500 \fi
3501 {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3502 \@namedef{bbl@ADJ@layout.tabular@on}{%
3503 \ifnum\bbl@tabular@mode=\tw@
3504 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3505 \else
3506 \chardef\bbl@tabular@mode\@ne
3507 \fi}
3508 \@namedef{bbl@ADJ@layout.tabular@off}{%
3509 \ifnum\bbl@tabular@mode=\tw@
3510 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3511 \else
3512 \chardef\bbl@tabular@mode\z@
3513 \fi}
3514 \@namedef{bbl@ADJ@layout.lists@on}{%
3515 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3516 \@namedef{bbl@ADJ@layout.lists@off}{%
3517 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3518 %
3519 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3520 \bbl@bcpallowedtrue}
3521 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3522 \bbl@bcpallowedfalse}
3523 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3524 \def\bbl@bcp@prefix{#1}}
3525 \def\bbl@bcp@prefix{bcp47-}
3526 \@namedef{bbl@ADJ@autoload.options}#1{%
3527 \def\bbl@autoload@options{#1}}
3528 \def\bbl@autoload@bcptoptions{import}
3529 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3530 \def\bbl@autoload@bcptoptions{#1}}
3531 \newif\ifbbl@bcptname
3532 %
3533 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3534 \bbl@bcptnametrue}
3535 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3536 \bbl@bcptnamefalse}
3537 %
3538 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3539 \directlua{ Babel.ignore_pre_char = function(node)
3540 return (node.lang == \the\csname l@nohyphenation\endcsname)
3541 end }}
3542 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3543 \directlua{ Babel.ignore_pre_char = function(node)
3544 return false
3545 end }}
3546 %
3547 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3548 \def\bbl@ignoreinterchar{%
3549 \ifnum\language=\l@nohyphenation
3550 \expandafter\@gobble

```

```

3551 \else
3552 \expandafter\@firstofone
3553 \fi}}
3554 \@namedef{bbl@ADJ@interchar.disable@off}{%
3555 \let\bbl@ignoreinterchar\@firstofone}
3556 %
3557 \@namedef{bbl@ADJ@select.write@shift}{%
3558 \let\bbl@restorelastskip\relax
3559 \def\bbl@savelastskip{%
3560 \let\bbl@restorelastskip\relax
3561 \ifvmode
3562 \ifdim\lastskip=\z@
3563 \let\bbl@restorelastskip\nobreak
3564 \else
3565 \bbl@exp{%
3566 \def\\bbl@restorelastskip{%
3567 \skip@=\the\lastskip
3568 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3569 \fi
3570 \fi}}
3571 \@namedef{bbl@ADJ@select.write@keep}{%
3572 \let\bbl@restorelastskip\relax
3573 \let\bbl@savelastskip\relax}
3574 \@namedef{bbl@ADJ@select.write@omit}{%
3575 \AddBabelHook{babel-select}{beforestart}{%
3576 \expandafter\babel@aux\expandafter\bbl@main@language{}}}%
3577 \let\bbl@restorelastskip\relax
3578 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3579 \@namedef{bbl@ADJ@select.encoding@off}{%
3580 \let\bbl@encoding@select@off\@empty}

```

## 5.1. Cross referencing macros

The  $\TeX$  book states:

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

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

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

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

```

3581 <<(*More package options)>> ≡
3582 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3583 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3584 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3585 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3586 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3587 <</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.

```

3588 \bbl@trace{Cross referencing macros}
3589 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3590 \def\@newl@bel#1#2#3{%
3591 {\@safe@activestrue
3592 \bbl@ifunset{#1@#2}%
3593 \relax
3594 {\gdef\@multiplelabels{%
3595 \@latex@warning@no@line{There were multiply-defined labels}}%

```



```

3596      \@latex@warning@no@line{Label `#2' multiply defined}}%
3597      \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.

```

3598 \CheckCommand*\@testdef[3]{%
3599   \def\reserved@a{#3}%
3600   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3601   \else
3602     \@tempswatrue
3603   \fi}

```

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

```

3604 \def\@testdef#1#2#3{%
3605   \@safe@activestrue
3606   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3607   \def\bbl@tempb{#3}%
3608   \@safe@activestrue
3609   \ifx\bbl@tempa\relax
3610   \else
3611     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3612   \fi
3613   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3614   \ifx\bbl@tempa\bbl@tempb
3615   \else
3616     \@tempswatrue
3617   \fi}
3618 \fi

```

## **\ref**

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

```

3619 \bbl@xin@{R}\bbl@opt@safe
3620 \ifin@
3621   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3622   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3623   {\expandafter\strip@prefix\meaning\ref}%
3624 \ifin@
3625   \bbl@redefine\@kernel@ref#1{%
3626     \@safe@activestrue\org@@kernel@ref{#1}\@safe@activestrue}
3627   \bbl@redefine\@kernel@pageref#1{%
3628     \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activestrue}
3629   \bbl@redefine\@kernel@sref#1{%
3630     \@safe@activestrue\org@@kernel@sref{#1}\@safe@activestrue}
3631   \bbl@redefine\@kernel@spageref#1{%
3632     \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activestrue}
3633   \else
3634     \bbl@redefineroobust\ref#1{%
3635       \@safe@activestrue\org@ref{#1}\@safe@activestrue}
3636     \bbl@redefineroobust\pageref#1{%
3637       \@safe@activestrue\org@pageref{#1}\@safe@activestrue}
3638   \fi
3639 \else
3640   \let\org@ref\ref
3641   \let\org@pageref\pageref
3642 \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.

```
3643 \bbl@xin@{B}\bbl@opt@safe
3644 \ifin@
3645 \bbl@redefine\@citex[#1]#2{%
3646   \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activesfalse
3647   \org@citex[#1]{\bbl@tempa}}
```

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

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

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

```
3648 \AtBeginDocument{%
3649   \ifpackageloaded{natbib}{%
3650     \def\@citex[#1][#2]#3{%
3651       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activesfalse
3652       \org@citex[#1][#2]{\bbl@tempa}}%
3653   }{}}
```

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

```
3654 \AtBeginDocument{%
3655   \ifpackageloaded{cite}{%
3656     \def\@citex[#1]#2{%
3657       \@safe@activetrue\org@citex[#1]{#2}\@safe@activesfalse}%
3658   }{}}
```

**\nocite** The macro \nocite which is used to instruct BiB<sub>T</sub><sub>X</sub> to extract uncited references from the database.

```
3659 \bbl@redefine\nocite#1{%
3660   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3661 \bbl@redefine\bibcite{%
3662   \bbl@cite@choice
3663   \bibcite}
```

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

```
3664 \def\bbl@bibcite#1#2{%
3665   \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.

```
3666 \def\bbl@cite@choice{%
3667   \global\let\bibcite\bbl@bibcite
3668   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3669   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3670   \global\let\bbl@cite@choice\relax}
```

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

```
3671 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3672 \bbl@redefine\@bibitem#1{%
3673   \@safe@activestruelorg@@bibitem{#1}\@safe@activesfalse}
3674 \else
3675   \let\org@nocite\nocite
3676   \let\org@@citex\@citex
3677   \let\org@babcite\babcite
3678   \let\org@@bibitem\@bibitem
3679 \fi
```

## 5.2. Layout

```
3680 \newcommand\BabelPatchSection[1]{%
3681   \@ifundefined{#1}{}{%
3682     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3683     \@namedef{#1}{%
3684       \@ifstar{\bbl@presec@s{#1}}%
3685       {\@dblarg{\bbl@presec@x{#1}}}}%
3686 \def\bbl@presec@x#1[#2]#3{%
3687   \bbl@exp{%
3688     \\select@language@x{\bbl@main@language}%
3689     \\bbl@cs{sspre@#1}%
3690     \\bbl@cs{ss@#1}%
3691     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3692     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3693     \\select@language@x{\language}}%
3694 \def\bbl@presec@s#1#2{%
3695   \bbl@exp{%
3696     \\select@language@x{\bbl@main@language}%
3697     \\bbl@cs{sspre@#1}%
3698     \\bbl@cs{ss@#1}*%
3699     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3700     \\select@language@x{\language}}%
3701 %
3702 \IfBabelLayout{sectioning}%
3703   {\BabelPatchSection{part}%
3704    \BabelPatchSection{chapter}%
3705    \BabelPatchSection{section}%
3706    \BabelPatchSection{subsection}%
3707    \BabelPatchSection{subsubsection}%
3708    \BabelPatchSection{paragraph}%
3709    \BabelPatchSection{subparagraph}%
3710    \def\babel@toc#1{%
3711      \select@language@x{\bbl@main@language}}}%
3712 \IfBabelLayout{captions}%
3713   {\BabelPatchSection{caption}}}
```

**\BabelFootnote** Footnotes.

```
3714 \bbl@trace{Footnotes}
3715 \ifnum\bbl@bidimode>\z@ % Any bidi=
3716 \def\bbl@footnote#1#2#3{%
3717   \@ifnextchar[%
3718     {\bbl@footnote@o{#1}{#2}{#3}}%
3719     {\bbl@footnote@x{#1}{#2}{#3}}}
3720 \long\def\bbl@footnote@x#1#2#3#4{%
3721   \bgroup
3722   \select@language@x{\bbl@main@language}%

```

```

3723     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3724     \egroup}
3725 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3726     \bgroup
3727     \select@language@x{\bbl@main@language}%
3728     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3729     \egroup}
3730 \def\bbl@footnotetext#1#2#3{%
3731     \@ifnextchar[%
3732         {\bbl@footnotetext@o{#1}{#2}{#3}}%
3733         {\bbl@footnotetext@x{#1}{#2}{#3}}}%
3734 \long\def\bbl@footnotetext@x#1#2#3#4{%
3735     \bgroup
3736     \select@language@x{\bbl@main@language}%
3737     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3738     \egroup}
3739 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3740     \bgroup
3741     \select@language@x{\bbl@main@language}%
3742     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3743     \egroup}
3744 \def\BabelFootnote#1#2#3#4{%
3745     \ifx\bbl@fn@footnote\@undefined
3746         \let\bbl@fn@footnote\footnote
3747     \fi
3748     \ifx\bbl@fn@footnotetext\@undefined
3749         \let\bbl@fn@footnotetext\footnotetext
3750     \fi
3751     \bbl@ifblank{#2}%
3752         {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3753          \namedef{\bbl@stripslash#1text}%
3754             {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3755         {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}{#3}{#4}}%
3756          \namedef{\bbl@stripslash#1text}%
3757             {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}%
3758 \fi
3759 \IfBabelLayout{footnotes}%
3760     {\let\bbl@OL@footnote\footnote
3761      \BabelFootnote\footnote\language\name{}}%
3762     {\BabelFootnote\localfootnote\language\name{}}%
3763     {\BabelFootnote\mainfootnote{}}{}%
3764     {}

```

### 5.3. Marks

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

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

```

3765 \bbl@trace{Marks}
3766 \IfBabelLayout{sectioning}
3767     {\ifx\bbl@opt@headfoot\@nnil
3768         \g@addto@macro\resetactivechars{%
3769             \set@typeset@protect
3770             \expandafter\select@language@x\expandafter{\bbl@main@language}%
3771             \let\protect\noexpand
3772             \ifcase\bbl@bidimode\else % Only with bidi. See also above
3773                 \edef\thepage{%
3774                     \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3775             \fi}%
3776     \fi}

```

```

3777 {\ifbbl@single\else
3778   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3779   \markright#1{%
3780     \bbl@ifblank{#1}%
3781     {\org@markright{}}}%
3782     {\toks@{#1}%
3783     \bbl@exp{%
3784       \\org@markright{\\protect\\foreignlanguage{\language}%
3785       {\\protect\\bbl@restore@actives\the\toks@}}}}}%

```

## **\markboth**

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

```

3786   \ifx\@mkboth\markboth
3787     \def\bbl@tempc{\let\@mkboth\markboth}%
3788   \else
3789     \def\bbl@tempc{%
3790   \fi
3791   \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3792   \markboth#1#2{%
3793     \protected@edef\bbl@tempb##1{%
3794       \protect\foreignlanguage
3795       {\language}{\protect\bbl@restore@actives##1}}%
3796     \bbl@ifblank{#1}%
3797     {\toks@{}}%
3798     {\toks@\expandafter{\bbl@tempb{#1}}}%
3799     \bbl@ifblank{#2}%
3800     {\@temptokena{}}%
3801     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3802     \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3803     \bbl@tempc
3804   \fi} % end ifbbl@single, end \IfBabelLayout

```

## 5.4. Other packages

### 5.4.1. ifthen

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

```

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

```

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

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

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

```

3805 \bbl@trace{Preventing clashes with other packages}
3806 \ifx\org@ref\undefined\else
3807   \bbl@xin@{R}\bbl@opt@safe
3808   \ifin@
3809     \AtBeginDocument{%

```

```

3810 \ifpackageloaded{ifthen}{%
3811 \bbl@redefine@long\ifthenelse#1#2#3{%
3812 \let\bbl@temp@pref\pageref
3813 \let\pageref\org@pageref
3814 \let\bbl@temp@ref\ref
3815 \let\ref\org@ref
3816 \@safe@activetrue
3817 \org@ifthenelse{#1}%
3818 {\let\pageref\bbl@temp@pref
3819 \let\ref\bbl@temp@ref
3820 \@safe@activesfalse
3821 #2}%
3822 {\let\pageref\bbl@temp@pref
3823 \let\ref\bbl@temp@ref
3824 \@safe@activesfalse
3825 #3}%
3826 }%
3827 }{}%
3828 }
3829 \fi

```

#### 5.4.2. varioref

**\@@vpageref**

**\vrefpagenum**

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

```

3830 \AtBeginDocument{%
3831 \ifpackageloaded{varioref}{%
3832 \bbl@redefine\@@vpageref#1[#2]#3{%
3833 \@safe@activetrue
3834 \org@@vpageref{#1}[#2]{#3}%
3835 \@safe@activesfalse}%
3836 \bbl@redefine\vrefpagenum#1#2{%
3837 \@safe@activetrue
3838 \org@vrefpagenum{#1}{#2}%
3839 \@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<sub>␣</sub> 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.

```

3840 \expandafter\def\csname Ref \endcsname#1{%
3841 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3842 }{}%
3843 }
3844 \fi

```

#### 5.4.3. 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.

```

3845 \AtEndOfPackage{%
3846 \AtBeginDocument{%
3847 \ifpackageloaded{hhline}%
3848 {\expandafter\ifx\csname normal@char\string\endcsname\relax
3849 \else

```

```

3850         \makeatletter
3851         \def\@currname{hhline}\input{hhline.sty}\makeatother
3852         \fi}%
3853     {}}}

```

**\substitutefontfamily** *Deprecated.* It creates an fd file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names. Use the tools provided by L<sup>A</sup>T<sub>E</sub>X (\DeclareFontFamilySubstitution).

```

3854 \def\substitutefontfamily#1#2#3{%
3855     \lowercase{\immediate\openout15=#1#2.fd\relax}%
3856     \immediate\writel5{%
3857         \string\ProvidesFile{#1#2.fd}%
3858         [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3859         \space generated font description file]^J
3860         \string\DeclareFontFamily{#1}{#2}{}}^J
3861         \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3862         \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3863         \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3864         \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3865         \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3866         \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3867         \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3868         \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3869     }%
3870     \closeout15
3871 }
3872 \@onlypreamble\substitutefontfamily

```

## 5.5. Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>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

```

3873 \bbl@trace{Encoding and fonts}
3874 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3875 \newcommand\BabelNonText{TS1,T3,TS3}
3876 \let\org@TeX\TeX
3877 \let\org@LaTeX\LaTeX
3878 \let\ensureascii\@firstofone
3879 \let\asciientcoding\@empty
3880 \AtBeginDocument{%
3881     \def\@elt#1{,#1,}%
3882     \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3883     \let\@elt\relax
3884     \let\bbl@tempb\@empty
3885     \def\bbl@tempc{OT1}%
3886     \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3887         \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3888     \bbl@foreach\bbl@tempa{%
3889         \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3890         \ifin@
3891             \def\bbl@tempb{#1}% Store last non-ascii
3892         \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3893             \ifin@else
3894                 \def\bbl@tempc{#1}% Store last ascii
3895             \fi
3896         \fi}%
3897     \ifx\bbl@tempb\@empty\else

```

```

3898 \bbl@xin@{,\cf@encoding,}%{\BabelNonASCII,\BabelNonText,}%
3899 \ifin@else
3900 \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3901 \fi
3902 \let\asciencoding\bbl@tempc
3903 \renewcommand\ensureasci[1]{%
3904   {\fontencoding\asciencoding}\selectfont#1}}%
3905 \DeclareTextCommandDefault{\TeX}{\ensureasci{\org@TeX}}%
3906 \DeclareTextCommandDefault{\LaTeX}{\ensureasci{\org@LaTeX}}%
3907 \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.

```

3908 \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.

```

3909 \AtBeginDocument{%
3910   \@ifpackageloaded{fontspec}%
3911   {\xdef\latinencoding{%
3912     \ifx\UTFencname\undefined
3913       EU\ifcase\bbl@engine\or2\or1\fi
3914     \else
3915       \UTFencname
3916     \fi}}%
3917   {\gdef\latinencoding{OT1}%
3918     \ifx\cf@encoding\bbl@t@one
3919       \xdef\latinencoding{\bbl@t@one}%
3920     \else
3921       \def\@elt#1{,#1,}%
3922       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3923       \let\@elt\relax
3924       \bbl@xin@{,T1,}\bbl@tempa
3925       \ifin@
3926         \xdef\latinencoding{\bbl@t@one}%
3927       \fi
3928     \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.

```

3929 \DeclareRobustCommand{\latintext}{%
3930   \fontencoding{\latinencoding}\selectfont
3931   \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.

```

3932 \ifx\@undefined\DeclareTextFontCommand
3933   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3934 \else
3935   \DeclareTextFontCommand{\textlatin}{\latintext}
3936 \fi

```

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

```

3937 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```



## 5.6. Basic bidi support

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

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

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

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

```

3938 \bbl@trace{Loading basic (internal) bidi support}
3939 \ifodd\bbl@engine
3940 \else % Any xe+lua bidi
3941   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3942     \bbl@error{bidi-only-lua}{}}{}%
3943     \let\bbl@beforeforeign\leavevmode
3944     \AtEndOfPackage{%
3945       \EnableBabelHook{babel-bidi}%
3946       \bbl@xebidipar}
3947   \fi\fi
3948   \def\bbl@loadxebidi#1{%
3949     \ifx\RTLfootnotetext\@undefined
3950       \AtEndOfPackage{%
3951         \EnableBabelHook{babel-bidi}%
3952         \ifx\fontspec\@undefined
3953           \usepackage{fontspec}% bidi needs fontspec
3954         \fi
3955         \usepackage#1{bidi}%
3956         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3957         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3958           \ifnum\@nameuse{\bbl@wdir\@languagename}=\tw@ % 'AL' bidi
3959             \bbl@digitsdotdash % So ignore in 'R' bidi
3960           \fi}}%
3961     \fi}
3962   \ifnum\bbl@bidimode>200 % Any xe bidi=
3963     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3964       \bbl@tentative{bidi=bidi}
3965       \bbl@loadxebidi{}
3966     \or
3967       \bbl@loadxebidi{[rldocument]}
3968     \or
3969       \bbl@loadxebidi{}
3970     \fi
3971   \fi
3972 \fi
3973 \ifnum\bbl@bidimode=\@ne % bidi=default
3974   \let\bbl@beforeforeign\leavevmode
3975   \ifodd\bbl@engine % lua
3976     \newattribute\bbl@attr@dir
3977     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3978     \bbl@exp{\output{\bodydir\pagedir\the\output}}
```

```

3979 \fi
3980 \AtEndOfPackage{%
3981   \EnableBabelHook{babel-bidi}% pdf/luax/xe
3982   \ifodd\bbbl@engine\else % pdf/xe
3983     \bbbl@xebidipar
3984   \fi}
3985 \fi

```

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

```

3986 \bbbl@trace{Macros to switch the text direction}
3987 \def\bbbl@alscripts{%
3988   ,Arabic,Syriac,Thaana,Hanifi_Rohingya,Hanifi,Sogdian,}
3989 \def\bbbl@rscripts{%
3990   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
3991   Hatran,Hebrew,Imperial_Aramaic,Inscriptional_Pahlavi,%
3992   Inscriptional_Parthian,Kharoshthi,Lydian,Mandaic,Manichaean,%
3993   Mende_Kikakui,Meroitic_Cursive,Meroitic_Hieroglyphs,Nabataean,%
3994   Nko,Old_Hungarian,Old_North_Arabian,Old_Sogdian,%
3995   Old_South_Arabian,Old_Turkic,Old_Uyghur,Palmyrene,Phoenician,%
3996   Psalter_Pahlavi,Samaritan,Yezidi,Mandaean,%
3997   Meroitic,N'Ko,Orkhon,Todhri}
3998 %
3999 \def\bbbl@provide@dirs#1{%
4000   \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
4001   \ifin@
4002     \global\bbbl@csarg\chardef{wdir@#1}\@ne
4003     \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
4004     \ifin@
4005       \global\bbbl@csarg\chardef{wdir@#1}\tw@
4006     \fi
4007   \else
4008     \global\bbbl@csarg\chardef{wdir@#1}\z@
4009   \fi
4010   \ifodd\bbbl@engine
4011     \bbbl@csarg\ifcase{wdir@#1}%
4012       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4013     \or
4014       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4015     \or
4016       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4017     \fi
4018   \fi}
4019 %
4020 \def\bbbl@switchdir{%
4021   \bbbl@ifunset{bbbl@sys@\language name}{\bbbl@provide@sys@\language name}}{%
4022   \bbbl@ifunset{bbbl@wdir@\language name}{\bbbl@provide@dirs@\language name}}{%
4023   \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}}%
4024 \def\bbbl@setdirs#1{%
4025   \ifcase\bbbl@select@type
4026     \bbbl@bodydir{#1}%
4027     \bbbl@pardir{#1}% <- Must precede \bbbl@texdir
4028   \fi
4029   \bbbl@texdir{#1}}
4030 \ifnum\bbbl@bidimode>\z@
4031   \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4032   \DisableBabelHook{babel-bidi}
4033 \fi

```

Now the engine-dependent macros.

```

4034 \ifodd\bbbl@engine % luatex=1
4035 \else % pdftex=0, xetex=2
4036   \newcount\bbbl@dirlevel

```

```

4037 \chardef\bbL@thetextdir\z@
4038 \chardef\bbL@thepardir\z@
4039 \def\bbL@textdir#1{%
4040   \ifcase#1\relax
4041     \chardef\bbL@thetextdir\z@
4042     \@nameuse{setlatin}%
4043     \bbL@textdir@i\beginL\endL
4044   \else
4045     \chardef\bbL@thetextdir\@ne
4046     \@nameuse{setnonlatin}%
4047     \bbL@textdir@i\beginR\endR
4048   \fi}
4049 \def\bbL@textdir@i#1#2{%
4050   \ifhmode
4051     \ifnum\currentgrouplevel>\z@
4052       \ifnum\currentgrouplevel=\bbL@dirlevel
4053         \bbL@error{multiple-bidi}{\}\}\}%
4054         \bgroup\aftergroup#2\aftergroup\egroup
4055       \else
4056         \ifcase\currentgrouptype\or % 0 bottom
4057           \aftergroup#2% 1 simple {}
4058         \or
4059           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4060         \or
4061           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4062         \or\or\or % vbox vtop align
4063         \or
4064           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4065         \or\or\or\or\or\or % output math disc insert vcent mathchoice
4066         \or
4067           \aftergroup#2% 14 \begingroup
4068         \else
4069           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4070         \fi
4071       \fi
4072       \bbL@dirlevel\currentgrouplevel
4073     \fi
4074     #1%
4075   \fi}
4076 \def\bbL@pardir#1{\chardef\bbL@thepardir#1\relax}
4077 \let\bbL@bodydir\@gobble
4078 \let\bbL@pagedir\@gobble
4079 \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).

```

4080 \def\bbL@xebidipar{%
4081   \let\bbL@xebidipar\relax
4082   \TeXeTstate\@ne
4083   \def\bbL@xeverypar{%
4084     \ifcase\bbL@thepardir
4085       \ifcase\bbL@thetextdir\else\beginR\fi
4086     \else
4087       {\setbox\z@\lastbox\beginR\box\z@}%
4088     \fi}%
4089   \AddToHook{para/begin}{\bbL@xeverypar}}
4090 \ifnum\bbL@bidimode>200 % Any xe bidi=
4091   \let\bbL@textdir@i\@gobbletwo
4092   \let\bbL@xebidipar\@empty
4093   \AddBabelHook{bidi}{foreign}{%
4094     \ifcase\bbL@thetextdir
4095       \BabelWrapText{\LR{##1}}}%

```

```

4096     \else
4097         \BabelWrapText{\RL{##1}}%
4098     \fi}
4099     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4100 \fi
4101 \fi

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

4102 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4103 \AtBeginDocument{%
4104     \ifx\pdfstringdefDisableCommands\undefined\else
4105         \ifx\pdfstringdefDisableCommands\relax\else
4106             \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4107         \fi
4108     \fi}

```

## 5.7. Local Language Configuration

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

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

```

4109 \bbl@trace{Local Language Configuration}
4110 \ifx\loadlocalcfg\undefined
4111     \ifpackagewith{babel}{noconfigs}%
4112         {\let\loadlocalcfg\@gobble}%
4113     {\def\loadlocalcfg#1{%
4114         \InputIfFileExists{#1.cfg}%
4115         {\typeout{*****^J%
4116                 * Local config file #1.cfg used^^J%
4117                 *}}%
4118         \@empty}}
4119 \fi

```

## 5.8. Language options

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

```

4120 \bbl@trace{Language options}
4121 \let\bbl@afterlang\relax
4122 \let\BabelModifiers\relax
4123 \let\bbl@loaded\@empty
4124 \def\bbl@load@language#1{%
4125     \InputIfFileExists{#1.ldf}%
4126     {\edef\bbl@loaded{\CurrentOption
4127         \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4128         \expandafter\let\expandafter\bbl@afterlang
4129             \csname\CurrentOption.ldf-h@k\endcsname
4130         \expandafter\let\expandafter\BabelModifiers
4131             \csname\bbl@mod@\CurrentOption\endcsname
4132         \bbl@exp{\AtBeginDocument{
4133             \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4134     {\IfFileExists{babel-#1.tex}%
4135         {\def\bbl@tempa{%
4136             .\There is a locale ini file for this language.\%
4137             If it's the main language, try adding `provide=*'\%
4138             to the babel package options}}%
4139         {\let\bbl@tempa\empty}%
4140         \bbl@error{unknown-package-option}{}}}}

```

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

```

4141 \def\bbl@try@load@lang#1#2#3{%
4142   \IfFileExists{\CurrentOption.ldf}%
4143     {\bbl@load@language{\CurrentOption}}%
4144     {\#1\bbl@load@language{\#2}\#3}}
4145 %
4146 \DeclareOption{friulian}{\bbl@try@load@lang{}{friulan}{}}
4147 \DeclareOption{hebrew}{%
4148   \ifcase\bbl@engine\or
4149     \bbl@error{only-pdftex-lang}{hebrew}{luatex}{}%
4150   \fi
4151   \input{rlbabel.def}%
4152   \bbl@load@language{hebrew}}
4153 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4154 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4155 % \DeclareOption{northernkurdish}{\bbl@try@load@lang{}{kurmanji}{}}
4156 \DeclareOption{polutonikogreek}{%
4157   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4158 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4159 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4160 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}
```

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

If the language has been set as metadata, read the info from the corresponding ini file and extract the babel name. Then added it as a package option at the end, so that it becomes the main language. The behavior of a metatag with a global language option is not well defined, so if there is not a main option we set here explicitly.

```

4161 \ifx\GetDocumentProperties\undefined\else
4162   \edef\bbl@metalang{\GetDocumentProperties{document/lang}}%
4163   \ifx\bbl@metalang\empty\else
4164     \begingroup
4165       \expandafter
4166       \bbl@bcpllookup\bbl@metalang-\empty-\empty-\empty@@
4167       \bbl@read@ini{\bbl@bcp}\m@ne
4168       \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4169       \ifx\bbl@opt@main\@nnil
4170         \global\let\bbl@opt@main\language
4171       \fi
4172       \bbl@info{Passing \language\space to babel}%
4173     \endgroup
4174   \fi
4175 \fi
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.

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

```

4190 \def\bbl@tempf{,}
4191 \bbl@foreach\@raw@classoptionslist{%
4192   \in@{=}{#1}%
4193   \ifin@else
4194     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4195   \fi}
4196 \ifx\bbl@opt@main\@nnil
4197   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4198     \let\bbl@tempb\@empty
4199     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4200     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4201     \bbl@foreach\bbl@tempb{%      \bbl@tempb is a reversed list
4202       \ifx\bbl@opt@main\@nnil % i.e., if not yet assigned
4203         \ifodd\bbl@iniflag % = *=
4204           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4205         \else % n +=
4206           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4207         \fi
4208       \fi}%
4209   \fi
4210 \else
4211   \ifx\bbl@metalang\@undefined\else\ifx\bbl@metalang\@empty\else
4212     \bbl@afterfi\expandafter\@gobble
4213   \fi\fi % except if explicit lang metatag:
4214   {\bbl@info{Main language set with 'main='. Except if you have\\%
4215     problems, prefer the default mechanism for setting\\%
4216     the main language, i.e., as the last declared.\\%
4217     Reported}}
4218 \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`).

```

4219 \ifx\bbl@opt@main\@nnil\else
4220   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4221   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4222 \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.

```

4223 \bbl@foreach\bbl@language@opts{%
4224   \def\bbl@tempa{#1}%
4225   \ifx\bbl@tempa\bbl@opt@main\else
4226     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4227       \bbl@ifunset{ds@#1}%
4228       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4229       {}%
4230     \else % + * (other = ini)
4231       \DeclareOption{#1}{%
4232         \bbl@ldfinit
4233         \babelprovide[@import]{#1}% %%%
4234         \bbl@afterldf}%
4235     \fi
4236   \fi}
4237 \bbl@foreach\bbl@tempf{%
4238   \def\bbl@tempa{#1}%
4239   \ifx\bbl@tempa\bbl@opt@main\else

```

```

4240 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4241 \bbl@ifunset{ds@#1}%
4242 {\IfFileExists{#1.ldf}%
4243 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4244 {}}%
4245 {}%
4246 \else % + * (other = ini)
4247 \IfFileExists{babel-#1.tex}%
4248 {\DeclareOption{#1}%
4249 \bbl@ldfinit
4250 \babelprovide[@import]{#1}% %%%
4251 \bbl@afterldf}}%
4252 {}%
4253 \fi
4254 \fi}

```

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

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

```

4255 \NewHook{babel/presets}
4256 \UseHook{babel/presets}
4257 \def\AfterBabelLanguage#1{%
4258 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}}
4259 \DeclareOption*{}
4260 \ProcessOptions*

```

This finished the second pass. Now the third one begins, which loads the main language set with the key main. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. With some options in provide, the package `luatexbase` is loaded (and immediately used), and therefore `\babelprovide` can't go inside a `\DeclareOption`; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate `\AfterBabelLanguage`.

```

4261 \bbl@trace{Option 'main'}
4262 \ifx\bbl@opt@main\@nnil
4263 \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4264 \let\bbl@tempc\@empty
4265 \edef\bbl@templ{\bbl@loaded,}
4266 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4267 \bbl@for\bbl@tempb\bbl@tempa{%
4268 \edef\bbl@tempd{\bbl@tempb,}%
4269 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4270 \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4271 \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
4272 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4273 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4274 \ifx\bbl@tempb\bbl@tempc\else
4275 \bbl@warning{%
4276 Last declared language option is '\bbl@tempc',\%
4277 but the last processed one was '\bbl@tempb'.\%
4278 The main language can't be set as both a global\%
4279 and a package option. Use 'main=\bbl@tempc' as\%
4280 option. Reported}
4281 \fi
4282 \else
4283 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4284 \bbl@ldfinit
4285 \let\CurrentOption\bbl@opt@main
4286 \bbl@exp{% \bbl@opt@provide = empty if *
4287 \\\babelprovide
4288 [\bbl@opt@provide,@import,main]% %%%
4289 {\bbl@opt@main}}%
4290 \bbl@afterldf

```

```

4291 \DeclareOption{\bbl@opt@main}{}
4292 \else % case 0,2 (main is ldf)
4293 \ifx\bbl@loadmain\relax
4294 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4295 \else
4296 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4297 \fi
4298 \ExecuteOptions{\bbl@opt@main}
4299 \@namedef{ds@\bbl@opt@main}{}%
4300 \fi
4301 \DeclareOption*{}
4302 \ProcessOptions*
4303 \fi
4304 \bbl@exp{%
4305 \\\AtBeginDocument{\bbl@usehooks@lang{/}{\begindocument}{}}}%
4306 \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.

4307 \ifx\bbl@main@language\undefined
4308 \bbl@info{%
4309 You haven't specified a language as a class or package\\%
4310 option. I'll load 'nil'. Reported}
4311 \bbl@load@language{nil}
4312 \fi
4313 </package>

```

## 6. The kernel of Babel

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

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

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

A proxy file for `switch.def`

```

4314 <{*kernel}
4315 \let\bbl@onlyswitch\@empty
4316 \input babel.def
4317 \let\bbl@onlyswitch\@undefined
4318 </kernel>

```

## 7. Error messages

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

```

4319 <{*errors}
4320 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4321 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4322 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4323 \catcode`\@=11 \catcode`\^=7
4324 %
4325 \ifx\MessageBreak\undefined
4326 \gdef\bbl@error@i#1#2{%
4327 \begingroup

```



```

4328 \newlinechar=`^^J
4329 \def\{^^J(babel) }%
4330 \errhelp{#2}\errmessage{\{#1}%
4331 \endgroup}
4332 \else
4333 \gdef\bbl@error@i#1#2{%
4334 \beginingroup
4335 \def\{\MessageBreak}%
4336 \PackageError{babel}{#1}{#2}%
4337 \endgroup}
4338 \fi
4339 \def\bbl@errmessage#1#2#3{%
4340 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4341 \bbl@error@i{#2}{#3}}%
4342 % Implicit #2#3#4:
4343 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4344 %
4345 \bbl@errmessage{not-yet-available}
4346 {Not yet available}%
4347 {Find an armchair, sit down and wait}
4348 \bbl@errmessage{bad-package-option}%
4349 {Bad option '#1=#2'. Either you have misspelled the\\%
4350 key or there is a previous setting of '#1'. Valid\\%
4351 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4352 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4353 {See the manual for further details.}
4354 \bbl@errmessage{base-on-the-fly}
4355 {For a language to be defined on the fly 'base'\\%
4356 is not enough, and the whole package must be\\%
4357 loaded. Either delete the 'base' option or\\%
4358 request the languages explicitly}%
4359 {See the manual for further details.}
4360 \bbl@errmessage{undefined-language}
4361 {You haven't defined the language '#1' yet.\\%
4362 Perhaps you misspelled it or your installation\\%
4363 is not complete}%
4364 {Your command will be ignored, type <return> to proceed}
4365 \bbl@errmessage{shorthand-is-off}
4366 {I can't declare a shorthand turned off (\string#2)}
4367 {Sorry, but you can't use shorthands which have been\\%
4368 turned off in the package options}
4369 \bbl@errmessage{not-a-shorthand}
4370 {The character '\string #1' should be made a shorthand character;\\%
4371 add the command \string\usesshorthands\string{#1\string} to
4372 the preamble.\\%
4373 I will ignore your instruction}%
4374 {You may proceed, but expect unexpected results}
4375 \bbl@errmessage{not-a-shorthand-b}
4376 {I can't switch '\string#2' on or off--not a shorthand}%
4377 {This character is not a shorthand. Maybe you made\\%
4378 a typing mistake? I will ignore your instruction.}
4379 \bbl@errmessage{unknown-attribute}
4380 {The attribute #2 is unknown for language #1.}%
4381 {Your command will be ignored, type <return> to proceed}
4382 \bbl@errmessage{missing-group}
4383 {Missing group for string \string#1}%
4384 {You must assign strings to some category, typically\\%
4385 captions or extras, but you set none}
4386 \bbl@errmessage{only-lua-xe}
4387 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4388 {Consider switching to these engines.}
4389 \bbl@errmessage{only-lua}
4390 {This macro is available only in LuaLaTeX}%

```

```

4391 {Consider switching to that engine.}
4392 \bbl@errmessage{unknown-provide-key}
4393 {Unknown key '#1' in \string\babelprovide}%
4394 {See the manual for valid keys}%
4395 \bbl@errmessage{unknown-mapfont}
4396 {Option '\bbl@KVP@mapfont' unknown for\\%
4397 mapfont. Use 'direction'}%
4398 {See the manual for details.}
4399 \bbl@errmessage{no-ini-file}
4400 {There is no ini file for the requested language\\%
4401 (#1: \language). Perhaps you misspelled it or your\\%
4402 installation is not complete}%
4403 {Fix the name or reinstall babel.}
4404 \bbl@errmessage{digits-is-reserved}
4405 {The counter name 'digits' is reserved for mapping\\%
4406 decimal digits}%
4407 {Use another name.}
4408 \bbl@errmessage{limit-two-digits}
4409 {Currently two-digit years are restricted to the\\
4410 range 0-9999}%
4411 {There is little you can do. Sorry.}
4412 \bbl@errmessage{alphabetic-too-large}
4413 {Alphabetic numeral too large (#1)}%
4414 {Currently this is the limit.}
4415 \bbl@errmessage{no-ini-info}
4416 {I've found no info for the current locale.\\%
4417 The corresponding ini file has not been loaded\\%
4418 Perhaps it doesn't exist}%
4419 {See the manual for details.}
4420 \bbl@errmessage{unknown-ini-field}
4421 {Unknown field '#1' in \string\BCPdata.\\%
4422 Perhaps you misspelled it}%
4423 {See the manual for details.}
4424 \bbl@errmessage{unknown-locale-key}
4425 {Unknown key for locale '#2':\\%
4426 #3\\%
4427 \string#1 will be set to \string\relax}%
4428 {Perhaps you misspelled it.}%
4429 \bbl@errmessage{adjust-only-vertical}
4430 {Currently, #1 related features can be adjusted only\\%
4431 in the main vertical list}%
4432 {Maybe things change in the future, but this is what it is.}
4433 \bbl@errmessage{layout-only-vertical}
4434 {Currently, layout related features can be adjusted only\\%
4435 in vertical mode}%
4436 {Maybe things change in the future, but this is what it is.}
4437 \bbl@errmessage{bidi-only-lua}
4438 {The bidi method 'basic' is available only in\\%
4439 luatex. I'll continue with 'bidi=default', so\\%
4440 expect wrong results}%
4441 {See the manual for further details.}
4442 \bbl@errmessage{multiple-bidi}
4443 {Multiple bidi settings inside a group}%
4444 {I'll insert a new group, but expect wrong results.}
4445 \bbl@errmessage{unknown-package-option}
4446 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4447 or the language definition file \CurrentOption.ldf\\%
4448 was not found%
4449 \bbl@tempa}
4450 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4451 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4452 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4453 \bbl@errmessage{config-not-found}

```

```

4454 {Local config file '\bbl@opt@config.cfg' not found}%
4455 {Perhaps you misspelled it.}
4456 \bbl@errmessage{late-after-babel}
4457 {Too late for \string\AfterBabelLanguage}%
4458 {Languages have been loaded, so I can do nothing}
4459 \bbl@errmessage{double-hyphens-class}
4460 {Double hyphens aren't allowed in \string\babelcharclass\\%
4461   because it's potentially ambiguous}%
4462 {See the manual for further info}
4463 \bbl@errmessage{unknown-interchar}
4464 {'#1' for '\language' cannot be enabled.\\%
4465   Maybe there is a typo}%
4466 {See the manual for further details.}
4467 \bbl@errmessage{unknown-interchar-b}
4468 {'#1' for '\language' cannot be disabled.\\%
4469   Maybe there is a typo}%
4470 {See the manual for further details.}
4471 \bbl@errmessage{charproperty-only-vertical}
4472 {\string\babelcharproperty\space can be used only in\\%
4473   vertical mode (preamble or between paragraphs)}%
4474 {See the manual for further info}
4475 \bbl@errmessage{unknown-char-property}
4476 {No property named '#2'. Allowed values are\\%
4477   direction (bc), mirror (bmg), and linebreak (lb)}%
4478 {See the manual for further info}
4479 \bbl@errmessage{bad-transform-option}
4480 {Bad option '#1' in a transform.\\%
4481   I'll ignore it but expect more errors}%
4482 {See the manual for further info.}
4483 \bbl@errmessage{font-conflict-transforms}
4484 {Transforms cannot be re-assigned to different\\%
4485   fonts. The conflict is in '\bbl@kv@label'.\\%
4486   Apply the same fonts or use a different label}%
4487 {See the manual for further details.}
4488 \bbl@errmessage{transform-not-available}
4489 {'#1' for '\language' cannot be enabled.\\%
4490   Maybe there is a typo or it's a font-dependent transform}%
4491 {See the manual for further details.}
4492 \bbl@errmessage{transform-not-available-b}
4493 {'#1' for '\language' cannot be disabled.\\%
4494   Maybe there is a typo or it's a font-dependent transform}%
4495 {See the manual for further details.}
4496 \bbl@errmessage{year-out-range}
4497 {Year out of range.\\%
4498   The allowed range is #1}%
4499 {See the manual for further details.}
4500 \bbl@errmessage{only-pdfTeX-lang}
4501 {The '#1' ldf style doesn't work with #2,\\%
4502   but you can use the ini locale instead.\\%
4503   Try adding 'provide=' to the option list. You may\\%
4504   also want to set 'bidi=' to some value}%
4505 {See the manual for further details.}
4506 \bbl@errmessage{hyphenmins-args}
4507 {\string\babelhyphenmins\ accepts either the optional\\%
4508   argument or the star, but not both at the same time}%
4509 {See the manual for further details.}
4510 </errors>
4511 <*patterns>

```

## 8. Loading hyphenation patterns

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

```
4512 <@Make sure ProvidesFile is defined>
4513 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4514 \xdef\bbl@format{\jobname}
4515 \def\bbl@version{<@version@>}
4516 \def\bbl@date{<@date@>}
4517 \ifx\AtBeginDocument\undefined
4518   \def\@empty{}
4519 \fi
4520 <@Define core switching macros@>
```

**\process@line** Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with `=`. When the first token of a line is an `=`, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

```
4521 \def\process@line#1#2 #3 #4 {%
4522   \ifx=#1%
4523     \process@synonym{#2}%
4524   \else
4525     \process@language{#1#2}{#3}{#4}%
4526   \fi
4527   \ignorespaces}
```

**\process@synonym** This macro takes care of the lines which start with an `=`. It needs an empty token register to begin with. `\bbl@languages` is also set to empty.

```
4528 \toks@{}
4529 \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.

```
4530 \def\process@synonym#1{%
4531   \ifnum\last@language=\m@ne
4532     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4533   \else
4534     \expandafter\chardef\csname l@#1\endcsname\last@language
4535     \wlog{\string\l@#1=\string\language\the\last@language}%
4536     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
4537     \csname\language\endcsname hyphenmins\endcsname
4538     \let\bbl@elt\relax
4539     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4540   \fi}
```

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

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

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

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

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

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

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

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

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

```

4541 \def\process@language#1#2#3{%
4542   \expandafter\addlanguage\csname l@#1\endcsname
4543   \expandafter\language\csname l@#1\endcsname
4544   \edef\language#1{%
4545     \bbl@hook@everylanguage{#1}%
4546     % > luatex
4547     \bbl@get@enc#1::\@@@
4548   \begingroup
4549     \lefthyphenmin\m@ne
4550     \bbl@hook@loadpatterns{#2}%
4551     % > luatex
4552     \ifnum\lefthyphenmin=\m@ne
4553     \else
4554       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4555         \the\lefthyphenmin\the\righthyphenmin}%
4556     \fi
4557   \endgroup
4558   \def\bbl@tempa{#3}%
4559   \ifx\bbl@tempa\@empty\else
4560     \bbl@hook@loadexceptions{#3}%
4561     % > luatex
4562   \fi
4563   \let\bbl@elt\relax
4564   \edef\bbl@languages{%
4565     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4566   \ifnum\the\language=\z@
4567     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4568       \set@hyphenmins\tw@\thr@@\relax
4569     \else
4570       \expandafter\expandafter\expandafter\set@hyphenmins
4571       \csname #1hyphenmins\endcsname
4572     \fi
4573     \the\toks@
4574     \toks@{}%
4575   \fi}

```

### **`\bbl@get@enc`**

**`\bbl@hyph@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.

```

4576 \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.

```

4577 \def\bbl@hook@everylanguage#1{}
4578 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4579 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4580 \def\bbl@hook@loadkernel#1{%
4581   \def\addlanguage{\csname newlanguage\endcsname}%

```

```

4582 \def\adddialect##1##2{%
4583   \global\chardef##1##2\relax
4584   \wlog{\string##1 = a dialect from \string\language##2}}%
4585 \def\iflanguage##1{%
4586   \expandafter\ifx\csname l@##1\endcsname\relax
4587     \@nolanner{##1}%
4588   \else
4589     \ifnum\csname l@##1\endcsname=\language
4590       \expandafter\expandafter\expandafter\@firstoftwo
4591     \else
4592       \expandafter\expandafter\expandafter\@secondoftwo
4593     \fi
4594   \fi}%
4595 \def\providehyphenmins##1##2{%
4596   \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4597     \@namedef{##1hyphenmins}{##2}%
4598   \fi}%
4599 \def\set@hyphenmins##1##2{%
4600   \lefthyphenmin##1\relax
4601   \righthyphenmin##2\relax}%
4602 \def\selectlanguage{%
4603   \errhelp{Selecting a language requires a package supporting it}%
4604   \errmessage{No multilingual package has been loaded}}%
4605 \let\foreignlanguage\selectlanguage
4606 \let\otherlanguage\selectlanguage
4607 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4608 \def\bbl@usehooks##1##2{%
4609 \def\setlocale{%
4610   \errhelp{Find an armchair, sit down and wait}%
4611   \errmessage{(babel) Not yet available}}%
4612 \let\uselocale\setlocale
4613 \let\locale\setlocale
4614 \let\selectlocale\setlocale
4615 \let\localename\setlocale
4616 \let\textlocale\setlocale
4617 \let\textlanguage\setlocale
4618 \let\languagetext\setlocale}
4619 \begingroup
4620 \def\AddBabelHook#1#2{%
4621   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4622     \def\next{\toks1}%
4623   \else
4624     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4625   \fi
4626   \next}
4627 \ifx\directlua\@undefined
4628   \ifx\XeTeXinputencoding\@undefined\else
4629     \input xebabel.def
4630   \fi
4631 \else
4632   \input luababel.def
4633 \fi
4634 \openin1 = babel-\bbl@format.cfg
4635 \ifeof1
4636 \else
4637   \input babel-\bbl@format.cfg\relax
4638 \fi
4639 \closein1
4640 \endgroup
4641 \bbl@hook@loadkernel{switch.def}

```

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

```

4642 \openin1 = language.dat

```

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

```
4643 \def\language{english}%
4644 \ifeof1
4645   \message{I couldn't find the file language.dat,\space
4646           I will try the file hyphen.tex}
4647   \input hyphen.tex\relax
4648   \chardef\l@english\z@
4649 \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`.

```
4650 \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.

```
4651 \loop
4652   \endlinechar@m@ne
4653   \read1 to \bbl@line
4654   \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.

```
4655   \if T\ifeof1F\fi T\relax
4656   \ifx\bbl@line\empty\else
4657     \edef\bbl@line{\bbl@line\space\space\space}%
4658     \expandafter\process@line\bbl@line\relax
4659   \fi
4660 \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.

```
4661 \begingroup
4662   \def\bbl@elt#1#2#3#4{%
4663     \global\language=#2\relax
4664     \gdef\language{#1}%
4665     \def\bbl@elt##1##2##3##4{}}%
4666   \bbl@languages
4667 \endgroup
4668 \fi
4669 \closein1
```

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

```
4670 \if\the\toks@\else
4671   \errhelp{language.dat loads no language, only synonyms}
4672   \errmessage{Orphan language synonym}
4673 \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.

```
4674 \let\bbl@line\@undefined
4675 \let\process@line\@undefined
4676 \let\process@synonym\@undefined
4677 \let\process@language\@undefined
4678 \let\bbl@get@enc\@undefined
4679 \let\bbl@hyph@enc\@undefined
4680 \let\bbl@tempa\@undefined
4681 \let\bbl@hook@loadkernel\@undefined
4682 \let\bbl@hook@everylanguage\@undefined
```

```

4683 \let\bbl@hook@loadpatterns\@undefined
4684 \let\bbl@hook@loadexceptions\@undefined
4685 \</patterns>

```

Here the code for `iniTeX` ends.

## 9. luatex + xetex: common stuff

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 (although default also applies to `pdfTeX`).

```

4686 <<*More package options>> ≡
4687 \chardef\bbl@bidimode\z@
4688 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4689 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4690 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4691 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4692 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4693 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4694 <</More package options>>

```

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

```

4695 <<*Font selection>> ≡
4696 \bbl@trace{Font handling with fontspec}
4697 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4698 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4699 \DisableBabelHook{babel-fontspec}
4700 \@onlypreamble\babelfont
4701 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4702   \ifx\fontspec\@undefined
4703     \usepackage{fontspec}%
4704     \fi
4705     \EnableBabelHook{babel-fontspec}%
4706     \edef\bbl@tempa{#1}%
4707     \def\bbl@tempb{#2}% Used by \bbl@bblfont
4708     \bbl@bblfont}
4709 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4710   \bbl@ifunset{\bbl@tempb family}%
4711     {\bbl@providefam{\bbl@tempb}}%
4712     {}%
4713   % For the default font, just in case:
4714   \bbl@ifunset{\bbl@sys\language}{\bbl@provide@sys{\language}}{}%
4715   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4716     {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
4717     \bbl@exp{%
4718       \let<bbl@\bbl@tempb dflt@\language>\bbl@\bbl@tempb dflt@>%
4719       \bbl@font@set<bbl@\bbl@tempb dflt@\language>%
4720       \<bbl@tempb default>\<bbl@tempb family>}}%
4721     {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4722       \bbl@csarg\def{\bbl@tempb dflt@##1}{<#1>{#2}}}%

```

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

```

4723 \def\bbl@providefam#1{%
4724   \bbl@exp{%
4725     \\\newcommand<#1default>{}% Just define it
4726     \\\bbl@add@list\\bbl@font@fams{#1}%
4727     \\\NewHook{#1family}%
4728     \\\DeclareRobustCommand<#1family>{%
4729       \\\not@math@alphabet<#1family>\relax
4730       % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails

```



```

4731     \\fontfamily\<#1default>%
4732     \\UseHook{#1family}%
4733     \\selectfont}%
4734     \\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.

```

4735 \def\bbl@nostdfont#1{%
4736   \bbl@ifunset{bbl@WFF@f@family}%
4737   {\bbl@csarg\gdef{WFF@f@family}{}}% Flag, to avoid dupl warns
4738   \bbl@infowarn{The current font is not a babel standard family:\\%
4739     #1%
4740     \fontname\font\\%
4741     There is nothing intrinsically wrong with this warning, and\\%
4742     you can ignore it altogether if you do not need these\\%
4743     families. But if they are used in the document, you should be\\%
4744     aware 'babel' will not set Script and Language for them, so\\%
4745     you may consider defining a new family with \string\babelfont.\\%
4746     See the manual for further details about \string\babelfont.\\%
4747     Reported}}
4748   {}}%
4749 \gdef\bbl@switchfont{%
4750   \bbl@ifunset{bbl@lsys@\language name}{\bbl@provide@lsys{\language name}}{%
4751     \bbl@exp{ e.g., Arabic -> arabic
4752       \lowercase{\edef\bbl@tempa{\bbl@ccl{sname}}}}%
4753     \bbl@foreach\bbl@font@fams{%
4754       \bbl@ifunset{bbl@##1dflt@\language name}% (1) language?
4755       {\bbl@ifunset{bbl@##1dflt*@\bbl@tempa}% (2) from script?
4756         {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4757           {}}% 123=F - nothing!
4758         {\bbl@exp{ 3=T - from generic
4759           \global\let\bbl@##1dflt@\language name}%
4760           \<bbl@##1dflt@>}}}%
4761         {\bbl@exp{ 2=T - from script
4762           \global\let\bbl@##1dflt@\language name}%
4763           \<bbl@##1dflt*@\bbl@tempa>}}}%
4764         {}}% 1=T - language, already defined
4765     \def\bbl@tempa{\bbl@nostdfont{}}%
4766     \bbl@foreach\bbl@font@fams{% don't gather with prev for
4767       \bbl@ifunset{bbl@##1dflt@\language name}%
4768       {\bbl@cs{famrst@##1}%
4769         \global\bbl@csarg\let{famrst@##1}\relax}%
4770       {\bbl@exp{ order is relevant.
4771         \\bbl@add\\originalTeX{%
4772         \\bbl@font@rst{\bbl@ccl{##1dflt}}%
4773         \<##1default>\<##1family>{##1}}%
4774         \\bbl@font@set{\<bbl@##1dflt@\language name>% the main part!
4775         \<##1default>\<##1family>}}}%
4776       \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.

```

4777 \ifx\fontfamily\undefined\else % if latex
4778 \ifcase\bbl@engine % if pdftex
4779 \let\bbl@cckckstdfonts\relax
4780 \else
4781 \def\bbl@cckckstdfonts{%
4782   \begingroup
4783   \global\let\bbl@cckckstdfonts\relax
4784   \let\bbl@tempa\empty
4785   \bbl@foreach\bbl@font@fams{%
4786     \bbl@ifunset{bbl@##1dflt@}%
4787     {\@nameuse{##1family}%
4788     \bbl@csarg\gdef{WFF@f@family}{}}% Flag

```

```

4789         \bbl@exp{\bbl@add\bbl@tempa{* \<#1family>= \f@family\\}%
4790         \space\space\fontname\font\\}%
4791         \bbl@csarg\xdef{##1dflt@}{\f@family}%
4792         \expandafter\xdef\csname ##1default\endcsname{\f@family}%
4793         {}}%
4794     \ifx\bbl@tempa\empty\else
4795         \bbl@infowarn{The following font families will use the default\\%
4796         settings for all or some languages:\\%
4797         \bbl@tempa
4798         There is nothing intrinsically wrong with it, but\\%
4799         'babel' will no set Script and Language, which could\\%
4800         be relevant in some languages. If your document uses\\%
4801         these families, consider redefining them with \string\babelfont.\\%
4802         Reported}%
4803     \fi
4804 \endgroup}
4805 \fi
4806 \fi

```

Now the macros defining the font with fontspec.

When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily `\bbl@mapselect` because `\selectfont` is called internally when a font is defined.

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

```

4807 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4808 \bbl@xin@{<>}{#1}%
4809 \ifin@
4810 \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4811 \fi
4812 \bbl@exp{%
4813 \def\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
4814 \bbl@ifsamestring{#2}{\f@family}%
4815 {\#3%
4816 \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}%
4817 \let\bbl@tempa\relax}%
4818 {}%

```

Loaded locally, which does its job, but very must be global. The problem is how. This actually defines a font predeclared with `\babelfont`, making sure Script and Language names are defined. If they are not, the corresponding data in the ini file is used. The font is actually set temporarily to get the family name (`\f@family`). There is also a hack because by default some replacements related to the bold series are sometimes assigned to the wrong font (see issue #92).

```

4819 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4820 \let\bbl@tempe\bbl@mapselect
4821 \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
4822 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}%
4823 \let\bbl@mapselect\relax
4824 \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
4825 \let#4\empty % Make sure \renewfontfamily is valid
4826 \bbl@set@renderer
4827 \bbl@exp{%
4828 \let\bbl@temp@pfam<\bbl@stripslash#4\space> e.g., '\rmfamily '
4829 \<keys_if_exist:nnf>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4830 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4831 \<keys_if_exist:nnf>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4832 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4833 \renewfontfamily\#4%
4834 [\bbl@cl{sys},% xetex removes unknown features :-{
4835 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi

```

```

4836      #2]}{#3}% i.e., \bbl@exp{...}{#3}
4837 \bbl@unset@renderer
4838 \begingroup
4839      #4%
4840      \xdef#1{\f@family}%      e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4841 \endgroup
4842 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4843 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4844 \ifin@
4845 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4846 \fi
4847 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4848 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4849 \ifin@
4850 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4851 \fi
4852 \let#4\bbl@temp@fam
4853 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4854 \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.

```

4855 \def\bbl@font@rst#1#2#3#4{%
4856 \bbl@ccarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4857 \def\bbl@font@fams{rm,sf,tt}
4858 <</Font selection>>

```

## 10. Hooks for XeTeX and LuaTeX

### 10.1. XeTeX

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

Now, the code.

```

4859 <*>xetex>
4860 \def\BabelStringsDefault{unicode}
4861 \let\xebbl@stop\relax
4862 \AddBabelHook{xetex}{encodedcommands}{%
4863 \def\bbl@tempa{#1}%
4864 \ifx\bbl@tempa\@empty
4865 \XeTeXinputencoding"bytes"%
4866 \else
4867 \XeTeXinputencoding"#1"%
4868 \fi
4869 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4870 \AddBabelHook{xetex}{stopcommands}{%
4871 \xebbl@stop
4872 \let\xebbl@stop\relax}
4873 \def\bbl@input@classes{% Used in CJK intraspaces
4874 \input{load-unicode-xetex-classes.tex}%
4875 \let\bbl@input@classes\relax}
4876 \def\bbl@intraspace#1 #2 #3\@{
4877 \bbl@ccarg\gdef{xeisp@\languagename}%
4878 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4879 \def\bbl@intrapenalty#1\@{
4880 \bbl@ccarg\gdef{xeipn@\languagename}%
4881 {\XeTeXlinebreakpenalty #1\relax}}
4882 \def\bbl@provide@intraspace{%
4883 \bbl@xin@{/s}{\bbl@cl{lncbrk}}%

```

```

4884 \ifin@else\bbl@xin@{/c}{/\bbl@c{l{lnbrk}}}\fi
4885 \ifin@
4886 \bbl@ifunset{bbl@intsp@language\name}{}%
4887 {\expandafter\ifx\csname bbl@intsp@language\name\endcsname\@empty\else
4888 \ifx\bbl@KVP@intraspace\@nnil
4889 \bbl@exp{%
4890 \\\bbl@intraspace\bbl@c{l{intsp}}\\@}%
4891 \fi
4892 \ifx\bbl@KVP@intrapenalty\@nnil
4893 \bbl@intrapenalty0\@@
4894 \fi
4895 \fi
4896 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4897 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4898 \fi
4899 \ifx\bbl@KVP@intrapenalty\@nnil\else
4900 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4901 \fi
4902 \bbl@exp{%
4903 \\\bbl@add<extras\language>{%
4904 \XeTeXlinebreaklocale "\bbl@c{l{tbc}}"%
4905 \<bbl@xeisp@language>%
4906 \<bbl@xeipn@language>}%
4907 \\\bbl@tglobal<extras\language>%
4908 \\\bbl@add<noextras\language>{%
4909 \XeTeXlinebreaklocale ""}%
4910 \\\bbl@tglobal<noextras\language>}%
4911 \ifx\bbl@ispace\@undefined
4912 \gdef\bbl@ispace{\bbl@c{l{xeisp}}}%
4913 \ifx\AtBeginDocument\@notprerr
4914 \expandafter\@secondoftwo % to execute right now
4915 \fi
4916 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4917 \fi}%
4918 \fi}
4919 \ifx\DisableBabelHook\@undefined\endinput\fi
4920 \let\bbl@set@renderer\relax
4921 \let\bbl@unset@renderer\relax
4922 <@Font selection@>
4923 \def\bbl@provide@extra#1{}

```

Hack for unhyphenated line breaking. See \bbl@provide@lsys in the common code.

```

4924 \def\bbl@xenohyph@d{%
4925 \bbl@ifset{bbl@prehc@language}%
4926 {\ifnum\hyphenchar\font=\defaultthyphenchar
4927 \iffontchar\font\bbl@c{l{prehc}}\relax
4928 \hyphenchar\font\bbl@c{l{prehc}}\relax
4929 \else\iffontchar\font"200B
4930 \hyphenchar\font"200B
4931 \else
4932 \bbl@warning
4933 {Neither 0 nor ZERO WIDTH SPACE are available\\%
4934 in the current font, and therefore the hyphen\\%
4935 will be printed. Try changing the fontspec's\\%
4936 'HyphenChar' to another value, but be aware\\%
4937 this setting is not safe (see the manual).\\%
4938 Reported}%
4939 \hyphenchar\font\defaultthyphenchar
4940 \fi\fi
4941 \fi}%
4942 {\hyphenchar\font\defaultthyphenchar}}

```

## 10.2. Support for interchar

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

```
4943 \ifnum\xe@alloc@intercharclass<\thr@@
4944 \xe@alloc@intercharclass\thr@@
4945 \fi
4946 \chardef\bbl@xe@class@default=\z@
4947 \chardef\bbl@xe@class@cjkideogram=\@ne
4948 \chardef\bbl@xe@class@cjkleftpunctuation=\tw@
4949 \chardef\bbl@xe@class@cjkrightpunctuation=\thr@@
4950 \chardef\bbl@xe@class@boundary=4095
4951 \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.

```
4952 \AddBabelHook{babel-interchar}{beforeextras}{%
4953 \@nameuse{\bbl@xechars@\language@}}
4954 \DisableBabelHook{babel-interchar}
4955 \protected\def\bbl@charclass#1{%
4956 \ifnum\count@<\z@
4957 \count@-\count@
4958 \loop
4959 \bbl@exp{%
4960 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4961 \XeTeXcharclass\count@ \bbl@tempc
4962 \ifnum\count@<`#1\relax
4963 \advance\count@\@ne
4964 \repeat
4965 \else
4966 \babel@savevariable{\XeTeXcharclass`#1}%
4967 \XeTeXcharclass`#1 \bbl@tempc
4968 \fi
4969 \count@`#1\relax}
```

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

```
4970 \newcommand\bbl@ifinterchar[1]{%
4971 \let\bbl@tempa\@gobble % Assume to ignore
4972 \edef\bbl@tempb{\zap@space#1 \@empty}%
4973 \ifx\bbl@KVP@interchar\@nnil\else
4974 \bbl@replace\bbl@KVP@interchar{ }{,}%
4975 \bbl@foreach\bbl@tempb{%
4976 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
4977 \ifin@
4978 \let\bbl@tempa\@firstofone
4979 \fi}%
4980 \fi
4981 \bbl@tempa}
4982 \newcommand\IfBabelIntercharT[2]{%
4983 \bbl@carg\bbl@add{\bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
4984 \newcommand\babelcharclass[3]{%
4985 \EnableBabelHook{babel-interchar}%
4986 \bbl@csarg\newXeTeXintercharclass{xe@class@#2@#1}%
4987 \def\bbl@tempb##1{%
4988 \ifx##1\@empty\else
4989 \ifx##1-%
4990 \bbl@upto
```

```

4991 \else
4992 \bbl@charclass{%
4993 \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4994 \fi
4995 \expandafter\bbl@tempb
4996 \fi}%
4997 \bbl@ifunset{bbl@xechars@#1}%
4998 {\toks@{%
4999 \babel@savevariable\XeTeXinterchartokenstate
5000 \XeTeXinterchartokenstate\@ne
5001 }}%
5002 {\toks@\expandafter\expandafter\expandafter{%
5003 \csname bbl@xechars@#1\endcsname}}%
5004 \bbl@csarg\edef{xechars@#1}{%
5005 \the\toks@
5006 \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
5007 \bbl@tempb#3\@empty}}
5008 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
5009 \protected\def\bbl@upto{%
5010 \ifnum\count@>\z@
5011 \advance\count@\@ne
5012 \count@-\count@
5013 \else\ifnum\count@=\z@
5014 \bbl@charclass{-}%
5015 \else
5016 \bbl@error{double-hyphens-class}{\count@}{\count@}%
5017 \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>@<language>`.

```

5018 \def\bbl@ignoreinterchar{%
5019 \ifnum\language=\l@nohyphenation
5020 \expandafter\@gobble
5021 \else
5022 \expandafter\@firstofone
5023 \fi}
5024 \newcommand\babelinterchar[5][]{%
5025 \let\bbl@kv@label\@empty
5026 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5027 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5028 {\bbl@ignoreinterchar{#5}}%
5029 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5030 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5031 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5032 \XeTeXinterchartoks
5033 \@nameuse{bbl@xeclasse@\bbl@tempa @#2}
5034 \bbl@ifunset{bbl@xeclasse@\bbl@tempa @#2}{\bbl@tempa @#2} %
5035 \@nameuse{bbl@xeclasse@\bbl@tempb @#2}
5036 \bbl@ifunset{bbl@xeclasse@\bbl@tempb @#2}{\bbl@tempb @#2} %
5037 = \expandafter{%
5038 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5039 \csname\zap@space bbl@xeinter@\bbl@kv@label
5040 @#3@#4@#2 \@empty\endcsname}}}}
5041 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5042 \bbl@ifunset{bbl@ic@#1@language}%
5043 {\bbl@error{unknown-interchar}{#1}{\count@}}%
5044 {\bbl@csarg\let{ic@#1@language}\@firstofone}}
5045 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5046 \bbl@ifunset{bbl@ic@#1@language}%
5047 {\bbl@error{unknown-interchar-b}{#1}{\count@}}%
5048 {\bbl@csarg\let{ic@#1@language}\@gobble}}
5049 </xetex>

```

## 10.3. Layout

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

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

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

```
5050 <*xetex | texxet>
5051 \providecommand\bbl@provide@intraspace{}
5052 \bbl@trace{Redefinitions for bidi layout}

Finish here if there is no layout.

5053 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5054 \IfBabelLayout{nopars}
5055 {}
5056 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5057 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5058 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5059 \ifnum\bbl@bidimode>\z@
5060 \IfBabelLayout{pars}
5061 {\def\@hangfrom#1{%
5062   \setbox\@tempboxa\hbox{{#1}}%
5063   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5064   \noindent\box\@tempboxa}
5065 \def\raggedright{%
5066   \let\\\@centercr
5067   \bbl@startskip\z@skip
5068   \@rightskip\@flushglue
5069   \bbl@endskip\@rightskip
5070   \parindent\z@
5071   \parfillskip\bbl@startskip}
5072 \def\raggedleft{%
5073   \let\\\@centercr
5074   \bbl@startskip\@flushglue
5075   \bbl@endskip\z@skip
5076   \parindent\z@
5077   \parfillskip\bbl@endskip}}
5078 {}
5079 \fi
5080 \IfBabelLayout{lists}
5081 {\bbl@sreplace\list
5082   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5083   \def\bbl@listleftmargin{%
5084     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5085   \ifcase\bbl@engine
5086     \def\labelenumii{}\theenumii{}% pdfTeX doesn't reverse ()
5087     \def\p@enumiii{\p@enumii}\theenumii{}%
5088     \fi
5089     \bbl@sreplace\@verbatim
5090     {\leftskip\@totalleftmargin}%
5091     {\bbl@startskip\textwidth
5092       \advance\bbl@startskip-\linewidth}%
5093     \bbl@sreplace\@verbatim
5094     {\rightskip\z@skip}%
5095     {\bbl@endskip\z@skip}}%
5096 {}
5097 \IfBabelLayout{contents}
5098 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5099   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5100 {}
5101 \IfBabelLayout{columns}
```

```

5102 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5103 \def\bbl@outputbox#1{%
5104   \hb@xt@\textwidth{%
5105     \hskip\columnwidth
5106     \hfil
5107     {\normalcolor\vrule \@width\columnseprule}%
5108     \hfil
5109     \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5110     \hskip-\textwidth
5111     \hb@xt@\columnwidth{\box\@outputbox \hss}%
5112     \hskip\columnsep
5113     \hskip\columnwidth}}}%
5114 {}

```

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.

```

5115 \IfBabelLayout{counters*}%
5116 {\bbl@add\bbl@opt@layout{.counters.}%
5117 \AddToHook{shipout/before}{%
5118   \let\bbl@tempa\babelsublr
5119   \let\babelsublr\@firstofone
5120   \let\bbl@save@thepage\thepage
5121   \protected@edef\thepage{\thepage}%
5122   \let\babelsublr\bbl@tempa}%
5123 \AddToHook{shipout/after}{%
5124   \let\thepage\bbl@save@thepage}}{}
5125 \IfBabelLayout{counters}%
5126 {\let\bbl@latinarabic=\@arabic
5127 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
5128 \let\bbl@asciroman=\@roman
5129 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5130 \let\bbl@asciiRoman=\@Roman
5131 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5132 \fi % end if layout
5133 \</xetex | texpet>

```

## 10.4. 8-bit TeX

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

```

5134 \< *texpet>
5135 \def\bbl@provide@extra#1{%
5136   % == auto-select encoding ==
5137   \ifx\bbl@encoding@select@off\@empty\else
5138     \bbl@ifunset{\bbl@encoding@#1}%
5139     {\def\@elt##1{,##1,}%
5140     \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5141     \count@\z@
5142     \bbl@foreach\bbl@tempe{%
5143       \def\bbl@tempd{##1}% Save last declared
5144       \advance\count@\@one}%
5145     \ifnum\count@>\@one % (1)
5146       \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5147       \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5148       \bbl@replace\bbl@tempa{ },}%
5149       \global\bbl@csarg\let{encoding@#1}\@empty
5150       \bbl@xin@{\, \bbl@tempd,}{, \bbl@tempa,}%
5151       \ifin@ \else % if main encoding included in ini, do nothing
5152         \let\bbl@tempb\relax
5153         \bbl@foreach\bbl@tempa{%
5154           \ifx\bbl@tempb\relax
5155             \bbl@xin@{,##1,}{, \bbl@tempb,}%
5156             \ifin@\def\bbl@tempb{##1}\fi

```



```

5157         \fi}%
5158     \ifx\bbbl@tempb\relax\else
5159         \bbbl@exp{%
5160             \global\<bbbl@add>\<bbbl@preextras@#1>{\<bbbl@encoding@#1>}%
5161             \gdef\<bbbl@encoding@#1>{%
5162                 \\babel@save\\f@encoding
5163                 \\bbbl@add\\originalTeX{\\selectfont}%
5164                 \\fontencoding{\bbbl@tempb}%
5165                 \\selectfont}}%
5166         \fi
5167     \fi
5168     \fi}%
5169     }%
5170 \fi}
5171 \</texxet>

```

## 10.5. LuaTeX

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

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

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

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

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

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

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

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

```

5172 \<!*luatex>
5173 \directlua{ Babel = Babel or {} } % DL2
5174 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5175 \bbbl@trace{Read language.dat}
5176 \ifx\bbbl@readstream\undefined
5177     \csname newread\endcsname\bbbl@readstream
5178 \fi
5179 \begingroup
5180     \toks@{}
5181     \count@ \z@ % 0=start, 1=0th, 2=normal
5182     \def\bbbl@process@line#1#2 #3 #4 {%
5183         \ifx=#1%
5184             \bbbl@process@synonym#2}%

```

```

5185 \else
5186 \bbl@process@language{#1#2}{#3}{#4}%
5187 \fi
5188 \ignorespaces}
5189 \def\bbl@manylang{%
5190 \ifnum\bbl@last>\@ne
5191 \bbl@info{Non-standard hyphenation setup}%
5192 \fi
5193 \let\bbl@manylang\relax}
5194 \def\bbl@process@language#1#2#3{%
5195 \ifcase\count@
5196 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5197 \or
5198 \count@\tw@
5199 \fi
5200 \ifnum\count@=\tw@
5201 \expandafter\addlanguage\csname l@#1\endcsname
5202 \language\allocationnumber
5203 \chardef\bbl@last\allocationnumber
5204 \bbl@manylang
5205 \let\bbl@elt\relax
5206 \xdef\bbl@languages{%
5207 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5208 \fi
5209 \the\toks@
5210 \toks@{}}
5211 \def\bbl@process@synonym@aux#1#2{%
5212 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5213 \let\bbl@elt\relax
5214 \xdef\bbl@languages{%
5215 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5216 \def\bbl@process@synonym#1{%
5217 \ifcase\count@
5218 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5219 \or
5220 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5221 \else
5222 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5223 \fi}
5224 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
5225 \chardef\l@english\z@
5226 \chardef\l@USenglish\z@
5227 \chardef\bbl@last\z@
5228 \global\@namedef{bbl@hyphendata@0}{{\hyphen.tex}}
5229 \gdef\bbl@languages{%
5230 \bbl@elt{english}{0}{\hyphen.tex}}%
5231 \bbl@elt{USenglish}{0}{}%
5232 \else
5233 \global\let\bbl@languages@format\bbl@languages
5234 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5235 \ifnum#2>\z@
5236 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5237 \fi}%
5238 \xdef\bbl@languages{\bbl@languages}%
5239 \fi
5240 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5241 \bbl@languages
5242 \openin\bbl@readstream=language.dat
5243 \ifeof\bbl@readstream
5244 \bbl@warning{I couldn't find language.dat. No additional\\%
5245 patterns loaded. Reported}%
5246 \else
5247 \loop

```

```

5248     \endlinechar\m@ne
5249     \read\bb@l@readstream to \bb@l@line
5250     \endlinechar`\^^M
5251     \if T\ifeof\bb@l@readstream F\fi T\relax
5252     \ifx\bb@l@line\@empty\else
5253         \edef\bb@l@line{\bb@l@line\space\space\space}%
5254         \expandafter\bb@l@process@line\bb@l@line\relax
5255     \fi
5256     \repeat
5257 \fi
5258 \closein\bb@l@readstream
5259 \endgroup
5260 \bb@l@trace{Macros for reading patterns files}
5261 \def\bb@l@get@enc#1:#2:#3\@@@{\def\bb@l@hyph@enc{#2}}
5262 \ifx\babelcatcodetablenum\@undefined
5263     \ifx\newcatcodetable\@undefined
5264         \def\babelcatcodetablenum{5211}
5265         \def\bb@l@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5266     \else
5267         \newcatcodetable\babelcatcodetablenum
5268         \newcatcodetable\bb@l@pattcodes
5269     \fi
5270 \else
5271     \def\bb@l@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5272 \fi
5273 \def\bb@l@luapatterns#1#2{%
5274     \bb@l@get@enc#1:.\@@@
5275     \setbox\z@\hbox\bgroup
5276     \beginingroup
5277         \savecatcodetable\babelcatcodetablenum\relax
5278         \initcatcodetable\bb@l@pattcodes\relax
5279         \catcodetable\bb@l@pattcodes\relax
5280         \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5281         \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5282         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5283         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5284         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5285         \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5286         \input #1\relax
5287         \catcodetable\babelcatcodetablenum\relax
5288     \endgroup
5289     \def\bb@l@tempa{#2}%
5290     \ifx\bb@l@tempa\@empty\else
5291         \input #2\relax
5292     \fi
5293 \egroup}%
5294 \def\bb@l@patterns@lua#1{%
5295     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5296         \csname l@#1\endcsname
5297         \edef\bb@l@tempa{#1}%
5298     \else
5299         \csname l@#1:\f@encoding\endcsname
5300         \edef\bb@l@tempa{#1:\f@encoding}%
5301     \fi\relax
5302     \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5303     \@ifundefined{bb@l@hyphendata@the\language}%
5304     {\def\bb@l@elt##1##2###4{%
5305         \ifnum##2=\csname l@bb@l@tempa\endcsname % #2=spanish, dutch:OT1...
5306         \def\bb@l@tempb{##3}%
5307         \ifx\bb@l@tempb\@empty\else % if not a synonymous
5308             \def\bb@l@tempc{{##3}{##4}}%
5309         \fi
5310         \bb@l@csarg\xdef{hyphendata@##2}{\bb@l@tempc}%

```

```

5311     \fi}%
5312     \bbl@languages
5313     \@ifundefined{bbl@hyphendata@the\language}%
5314     {\bbl@info{No hyphenation patterns were set for\%
5315       language '\bbl@tempa'. Reported}}}%
5316     {\expandafter\expandafter\expandafter\bbl@luapatterns
5317       \csname bbl@hyphendata@the\language\endcsname}}{}
5318 \endinput\fi

```

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

```

5319 \ifx\DisableBabelHook\@undefined
5320   \AddBabelHook{luatex}{everylanguage}{%
5321     \def\process@language##1##2##3{%
5322       \def\process@line####1####2 ####3 ####4 {}}}
5323   \AddBabelHook{luatex}{loadpatterns}{%
5324     \input #1\relax
5325     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5326       {{#1}}}}
5327   \AddBabelHook{luatex}{loadexceptions}{%
5328     \input #1\relax
5329     \def\bbl@tempb##1##2{{##1}{#1}}%
5330     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5331       {\expandafter\expandafter\expandafter\bbl@tempb
5332         \csname bbl@hyphendata@the\language\endcsname}}
5333 \endinput\fi

```

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

```

5334 \begingroup
5335 \catcode`\%=12
5336 \catcode`\'=12
5337 \catcode`\|=12
5338 \catcode`\:=12
5339 \directlua{
5340   Babel.locale_props = Babel.locale_props or {}
5341   function Babel.lua_error(e, a)
5342     tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5343       e .. '}' .. (a or '') .. '}'{}])
5344   end
5345
5346   function Babel.bytes(line)
5347     return line:gsub(".",
5348       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5349   end
5350
5351   function Babel.begin_process_input()
5352     if luatexbase and luatexbase.add_to_callback then
5353       luatexbase.add_to_callback('process_input_buffer',
5354         Babel.bytes, 'Babel.bytes')
5355     else
5356       Babel.callback = callback.find('process_input_buffer')
5357       callback.register('process_input_buffer', Babel.bytes)
5358     end
5359   end
5360   function Babel.end_process_input ()
5361     if luatexbase and luatexbase.remove_from_callback then
5362       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5363     else
5364       callback.register('process_input_buffer', Babel.callback)
5365     end
5366   end
5367
5368   function Babel.str_to_nodes(fn, matches, base)
5369     local n, head, last

```

```

5370     if fn == nil then return nil end
5371     for s in string.utfvalues(fn(matches)) do
5372         if base.id == 7 then
5373             base = base.replace
5374         end
5375         n = node.copy(base)
5376         n.char = s
5377         if not head then
5378             head = n
5379         else
5380             last.next = n
5381         end
5382         last = n
5383     end
5384     return head
5385 end
5386
5387 Babel.linebreaking = Babel.linebreaking or {}
5388 Babel.linebreaking.before = {}
5389 Babel.linebreaking.after = {}
5390 Babel.locale = {}
5391 function Babel.linebreaking.add_before(func, pos)
5392     tex.print([[\\noexpand\\csname bbl@luahyphenate\\endcsname]])
5393     if pos == nil then
5394         table.insert(Babel.linebreaking.before, func)
5395     else
5396         table.insert(Babel.linebreaking.before, pos, func)
5397     end
5398 end
5399 function Babel.linebreaking.add_after(func)
5400     tex.print([[\\noexpand\\csname bbl@luahyphenate\\endcsname]])
5401     table.insert(Babel.linebreaking.after, func)
5402 end
5403
5404 function Babel.addpatterns(pp, lg)
5405     local lg = lang.new(lg)
5406     local pats = lang.patterns(lg) or ''
5407     lang.clear_patterns(lg)
5408     for p in pp:gmatch('[^%s]+') do
5409         ss = ''
5410         for i in string.utfcharacters(p:gsub('%d', '')) do
5411             ss = ss .. '%d?' .. i
5412         end
5413         ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5414         ss = ss:gsub('%%.%d%?$', '%%.')
5415         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5416         if n == 0 then
5417             tex.sprint(
5418                 [[\\string\\csname\\space bbl@info\\endcsname{New pattern: }]]
5419                 .. p .. [[]])
5420             pats = pats .. ' ' .. p
5421         else
5422             tex.sprint(
5423                 [[\\string\\csname\\space bbl@info\\endcsname{Renew pattern: }]]
5424                 .. p .. [[]])
5425         end
5426     end
5427     lang.patterns(lg, pats)
5428 end
5429
5430 Babel.characters = Babel.characters or {}
5431 Babel.ranges = Babel.ranges or {}
5432 function Babel.hlist_has_bidi(head)

```

```

5433     local has_bidi = false
5434     local ranges = Babel.ranges
5435     for item in node.traverse(head) do
5436         if item.id == node.id'glyph' then
5437             local itemchar = item.char
5438             local chardata = Babel.characters[itemchar]
5439             local dir = chardata and chardata.d or nil
5440             if not dir then
5441                 for nn, et in ipairs(ranges) do
5442                     if itemchar < et[1] then
5443                         break
5444                     elseif itemchar <= et[2] then
5445                         dir = et[3]
5446                         break
5447                     end
5448                 end
5449             end
5450             if dir and (dir == 'al' or dir == 'r') then
5451                 has_bidi = true
5452             end
5453         end
5454     end
5455     return has_bidi
5456 end
5457 function Babel.set_chranges_b (script, chrng)
5458     if chrng == '' then return end
5459     texio.write('Replacing ' .. script .. ' script ranges')
5460     Babel.script_blocks[script] = {}
5461     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.)%s') do
5462         table.insert(
5463             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5464     end
5465 end
5466
5467 function Babel.discard_sublr(str)
5468     if str:find( [[\string\indexentry]] ) and
5469        str:find( [[\string\babelsublr]] ) then
5470         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5471             function(m) return m:sub(2,-2) end )
5472     end
5473     return str
5474 end
5475 }
5476 \endgroup
5477 \ifx\newattribute\@undefined\else % Test for plain
5478     \newattribute\bbl@attr@locale % DL4
5479     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5480     \AddBabelHook{luatex}{beforeextras}{%
5481         \setattribute\bbl@attr@locale\localeid}
5482 \fi
5483 %
5484 \def\BabelStringsDefault{unicode}
5485 \let\luabbl@stop\relax
5486 \AddBabelHook{luatex}{encodedcommands}{%
5487     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5488     \ifx\bbl@tempa\bbl@tempb\else
5489         \directlua{Babel.begin_process_input()}%
5490         \def\luabbl@stop{%
5491             \directlua{Babel.end_process_input()}}%
5492     \fi}%
5493 \AddBabelHook{luatex}{stopcommands}{%
5494     \luabbl@stop
5495     \let\luabbl@stop\relax}

```

```

5496 %
5497 \AddBabelHook{luatex}{patterns}{%
5498   \ifundefined{bbl@hyphendata@the\language}%
5499     {\def\bbl@elt##1##2##3##4{%
5500       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5501       \def\bbl@tempb{##3}%
5502       \ifx\bbl@tempb\@empty\else % if not a synonymous
5503         \def\bbl@tempc{##3}{##4}%
5504       \fi
5505       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5506     \fi}%
5507   \bbl@languages
5508   \ifundefined{bbl@hyphendata@the\language}%
5509     {\bbl@info{No hyphenation patterns were set for\%
5510       language '#2'. Reported}}%
5511     {\expandafter\expandafter\expandafter\bbl@luapatterns
5512      \csname bbl@hyphendata@the\language\endcsname}}}%
5513 \ifundefined{bbl@patterns@}{}%
5514 \beginingroup
5515   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5516   \ifin@else
5517     \ifx\bbl@patterns@\@empty\else
5518       \directlua{ Babel.addpatterns(
5519         [[\bbl@patterns@]], \number\language) }%
5520     \fi
5521     \ifundefined{bbl@patterns@#1}%
5522       \@empty
5523       {\directlua{ Babel.addpatterns(
5524         [[\space\csname bbl@patterns@#1\endcsname]],
5525         \number\language) }}%
5526       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5527     \fi
5528   \endgroup}%
5529 \bbl@exp{%
5530   \bbl@ifunset{bbl@prehc@languagename}{}%
5531   {\bbl@ifblank{\bbl@cs{prehc@languagename}}}%
5532   {\prehyphenchar=\bbl@cl{prehc}\relax}}%

```

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

```

5533 \onlypreamble\babelpatterns
5534 \AtEndOfPackage{%
5535   \newcommand\babelpatterns[2][\@empty]{%
5536     \ifx\bbl@patterns\relax
5537       \let\bbl@patterns@\@empty
5538     \fi
5539     \ifx\bbl@pttnlist\@empty\else
5540       \bbl@warning{%
5541         You must not intermingle \string\selectlanguage\space and\%
5542         \string\babelpatterns\space or some patterns will not\%
5543         be taken into account. Reported}%
5544       \fi
5545     \ifx\@empty#1%
5546       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5547     \else
5548       \edef\bbl@tempb{\zap@space#1 \@empty}%
5549       \bbl@for\bbl@tempa\bbl@tempb{%
5550         \bbl@fixname\bbl@tempa
5551         \bbl@iflanguage\bbl@tempa{%
5552           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5553             \ifundefined{bbl@patterns@\bbl@tempa}%
5554               \@empty

```

```

5555         {\csname bbl@patterns@bbl@tempa\endcsname\space}%
5556         #2}}}%
5557     \fi}}

```

## 10.6. Southeast Asian scripts

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

Replace regular (i.e., 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.

```

5558 \def\bbl@intraspace#1 #2 #3\@{%
5559   \directlua{
5560     Babel.intraspaces = Babel.intraspaces or {}
5561     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5562       {b = #1, p = #2, m = #3}
5563     Babel.locale_props[\the\localeid].intraspace = %
5564       {b = #1, p = #2, m = #3}
5565   }}
5566 \def\bbl@intrapenalty#1\@{%
5567   \directlua{
5568     Babel.intrapenalties = Babel.intrapenalties or {}
5569     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5570     Babel.locale_props[\the\localeid].intrapenalty = #1
5571   }}
5572 \begingroup
5573 \catcode`\%=12
5574 \catcode`\&=14
5575 \catcode`\'=12
5576 \catcode`\~=12
5577 \gdef\bbl@seaintraspace{&
5578   \let\bbl@seaintraspace\relax
5579   \directlua{
5580     Babel.sea_enabled = true
5581     Babel.sea_ranges = Babel.sea_ranges or {}
5582     function Babel.set_chranges (script, chrng)
5583       local c = 0
5584       for s, e in string.gmatch(chrng..' ', '(.-%.%.(.-%s)') do
5585         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5586         c = c + 1
5587       end
5588     end
5589     function Babel.sea_disc_to_space (head)
5590       local sea_ranges = Babel.sea_ranges
5591       local last_char = nil
5592       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5593       for item in node.traverse(head) do
5594         local i = item.id
5595         if i == node.id'glyph' then
5596           last_char = item
5597         elseif i == 7 and item.subtype == 3 and last_char
5598           and last_char.char > 0x0C99 then
5599           quad = font.getfont(last_char.font).size
5600           for lg, rg in pairs(sea_ranges) do
5601             if last_char.char > rg[1] and last_char.char < rg[2] then
5602               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5603               local intraspace = Babel.intraspaces[lg]
5604               local intrapenalty = Babel.intrapenalties[lg]
5605               local n
5606               if intrapenalty ~= 0 then
5607                 n = node.new(14, 0)      &% penalty
5608                 n.penalty = intrapenalty
5609                 node.insert_before(head, item, n)
5610               end

```



```

5611         n = node.new(12, 13)      &% (glue, spaceskip)
5612         node.setglue(n, intraspace.b * quad,
5613                     intraspace.p * quad,
5614                     intraspace.m * quad)
5615         node.insert_before(head, item, n)
5616         node.remove(head, item)
5617     end
5618 end
5619 end
5620 end
5621 end
5622 }&
5623 \bbl@luahyphenate}

```

## 10.7. CJK line breaking

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

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

```

5624 \catcode\%=14
5625 \gdef\bbl@cjk intraspace{%
5626   \let\bbl@cjk intraspace\relax
5627   \directlua{
5628     require('babel-data-cjk.lua')
5629     Babel.cjk_enabled = true
5630     function Babel.cjk_linebreak(head)
5631       local GLYPH = node.id'glyph'
5632       local last_char = nil
5633       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5634       local last_class = nil
5635       local last_lang = nil
5636       for item in node.traverse(head) do
5637         if item.id == GLYPH then
5638           local lang = item.lang
5639           local LOCALE = node.get_attribute(item,
5640             Babel.attr_locale)
5641           local props = Babel.locale_props[LOCALE] or {}
5642           local class = Babel.cjk_class[item.char].c
5643           if props.cjk_quotes and props.cjk_quotes[item.char] then
5644             class = props.cjk_quotes[item.char]
5645           end
5646           if class == 'cp' then class = 'cl' % ) as CL
5647           elseif class == 'id' then class = 'I'
5648           elseif class == 'cj' then class = 'I' % loose
5649           end
5650           local br = 0
5651           if class and last_class and Babel.cjk_breaks[last_class][class] then
5652             br = Babel.cjk_breaks[last_class][class]
5653           end
5654           if br == 1 and props.linebreak == 'c' and
5655             lang ~= \the\l@nohyphenation\space and
5656             last_lang ~= \the\l@nohyphenation then
5657             local intrapenalty = props.intrapenalty
5658             if intrapenalty ~= 0 then
5659               local n = node.new(14, 0)      % penalty
5660               n.penalty = intrapenalty
5661               node.insert_before(head, item, n)
5662             end
5663             local intraspace = props.intraspace
5664             local n = node.new(12, 13)      % (glue, spaceskip)

```

```

5665         node.setglue(n, intraspace.b * quad,
5666                     intraspace.p * quad,
5667                     intraspace.m * quad)
5668         node.insert_before(head, item, n)
5669     end
5670     if font.getfont(item.font) then
5671         quad = font.getfont(item.font).size
5672     end
5673     last_class = class
5674     last_lang = lang
5675     else % if penalty, glue or anything else
5676         last_class = nil
5677     end
5678 end
5679 lang.hyphenate(head)
5680 end
5681 }%
5682 \bbl@lua-hyphenate}
5683 \gdef\bbl@lua-hyphenate{%
5684 \let\bbl@lua-hyphenate\relax
5685 \directlua{
5686   luatexbase.add_to_callback('hyphenate',
5687   function (head, tail)
5688     if Babel.linebreaking.before then
5689       for k, func in ipairs(Babel.linebreaking.before) do
5690         func(head)
5691       end
5692     end
5693     lang.hyphenate(head)
5694     if Babel.cjk_enabled then
5695       Babel.cjk_linebreak(head)
5696     end
5697     if Babel.linebreaking.after then
5698       for k, func in ipairs(Babel.linebreaking.after) do
5699         func(head)
5700       end
5701     end
5702     if Babel.set_hboxed then
5703       Babel.set_hboxed(head)
5704     end
5705     if Babel.sea_enabled then
5706       Babel.sea_disc_to_space(head)
5707     end
5708   end,
5709   'Babel.hyphenate')
5710 }}
5711 \endgroup
5712 %
5713 \def\bbl@provide@intraspace{%
5714   \bbl@ifunset{\bbl@intsp@{language}}{%
5715     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5716       \bbl@xin@{c}{\bbl@c{lnbrk}}}%
5717     \ifin@           % cjk
5718       \bbl@cjk-intraspace
5719       \directlua{
5720         Babel.locale_props = Babel.locale_props or {}
5721         Babel.locale_props[\the\localeid].linebreak = 'c'
5722       }%
5723       \bbl@exp{\\bbl@intraspace\bbl@c{intsp}}\\@@}%
5724       \ifx\bbl@KVP@intrapenalty\@nnil
5725         \bbl@intrapenalty0@@
5726       \fi
5727     \else           % sea

```

```

5728      \bbl@seaintraspace
5729      \bbl@exp{\bbl@intraspace\bbl@ccl{intsp}}\bbl@@@%
5730      \directlua{
5731          Babel.sea_ranges = Babel.sea_ranges or {}
5732          Babel.set_chranges('\bbl@ccl{sbcpr}',
5733                          '\bbl@ccl{chrng}')
5734      }%
5735      \ifx\bbl@KVP@intrapenalty\@nnil
5736          \bbl@intrapenalty0\@@
5737      \fi
5738      \fi
5739      \fi
5740      \ifx\bbl@KVP@intrapenalty\@nnil\else
5741          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5742      \fi}}

```

## 10.8. Arabic justification

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

```

5743 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5744 \def\bblar@chars{%
5745     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5746     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5747     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5748 \def\bblar@elongated{%
5749     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5750     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5751     0649,064A}
5752 \begingroup
5753     \catcode`_ =11 \catcode`\_:=11
5754     \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5755 \endgroup
5756 \gdef\bbl@arabicjust{%
5757     \let\bbl@arabicjust\relax
5758     \newattribute\bblar@kashida
5759     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5760     \bblar@kashida=\z@
5761     \bbl@patchfont{\bbl@parsejalt}}%
5762     \directlua{
5763         Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5764         Babel.arabic.elong_map[\the\localeid] = {}
5765         luatexbase.add_to_callback('post_linebreak_filter',
5766             Babel.arabic.justify, 'Babel.arabic.justify')
5767         luatexbase.add_to_callback('hpack_filter',
5768             Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5769     }%

```

Save both node lists to make replacement.

```

5770 \def\bblar@fetchjalt#1#2#3#4{%
5771     \bbl@exp{\bbl@foreach{#1}}{%
5772         \bbl@ifunset\bblar@JE@##1{%
5773             {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5774             {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5775         \directlua{%
5776             local last = nil
5777             for item in node.traverse(tex.box[0].head) do
5778                 if item.id == node.id'glyph' and item.char > 0x600 and
5779                     not (item.char == 0x200D) then
5780                     last = item
5781                 end
5782             end
5783             Babel.arabic.#3['##1#4'] = last.char

```

5784     }}}

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

```
5785 \gdef\bbl@parsejalt{%
5786   \ifx\addfontfeature\undefined\else
5787     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}}%
5788   \ifin@
5789     \directlua{%
5790       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5791         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5792         tex.print([[string\cswb\space bbl@parsejalti\endcswb]])
5793       end
5794     }%
5795   \fi
5796 \fi}
5797 \gdef\bbl@parsejalti{%
5798   \begingroup
5799     \let\bbl@parsejalt\relax % To avoid infinite loop
5800     \edef\bbl@tempb{\fontid\font}%
5801     \bblar@nofswarn
5802     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5803     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5804     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5805     \addfontfeature{RawFeature+=jalt}%
5806     % \@namedef{\bblar@JE@0643}{06AA}% todo: catch medial kaf
5807     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5808     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5809     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5810     \directlua{%
5811       for k, v in pairs(Babel.arabic.from) do
5812         if Babel.arabic.dest[k] and
5813           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5814           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5815             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5816         end
5817       end
5818     }%
5819   \endgroup}
```

The actual justification (inspired by CHICKENIZE).

```
5820 \begingroup
5821 \catcode`#=11
5822 \catcode`~=11
5823 \directlua{
5824
5825 Babel.arabic = Babel.arabic or {}
5826 Babel.arabic.from = {}
5827 Babel.arabic.dest = {}
5828 Babel.arabic.justify_factor = 0.95
5829 Babel.arabic.justify_enabled = true
5830 Babel.arabic.kashida_limit = -1
5831
5832 function Babel.arabic.justify(head)
5833   if not Babel.arabic.justify_enabled then return head end
5834   for line in node.traverse_id(node.id'hlist', head) do
5835     Babel.arabic.justify_hlist(head, line)
5836   end
5837   return head
5838 end
5839
5840 function Babel.arabic.justify_hbox(head, gc, size, pack)
5841   local has_inf = false
5842   if Babel.arabic.justify_enabled and pack == 'exactly' then
```

```

5843     for n in node.traverse_id(12, head) do
5844         if n.stretch_order > 0 then has_inf = true end
5845     end
5846     if not has_inf then
5847         Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5848     end
5849 end
5850 return head
5851 end
5852
5853 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5854     local d, new
5855     local k_list, k_item, pos_inline
5856     local width, width_new, full, k_curr, wt_pos, goal, shift
5857     local subst_done = false
5858     local elong_map = Babel.arabic.elong_map
5859     local cnt
5860     local last_line
5861     local GLYPH = node.id'glyph'
5862     local KASHIDA = Babel.attr_kashida
5863     local LOCALE = Babel.attr_locale
5864
5865     if line == nil then
5866         line = {}
5867         line.glue_sign = 1
5868         line.glue_order = 0
5869         line.head = head
5870         line.shift = 0
5871         line.width = size
5872     end
5873
5874     % Exclude last line. todo. But-- it discards one-word lines, too!
5875     % ? Look for glue = 12:15
5876     if (line.glue_sign == 1 and line.glue_order == 0) then
5877         elongs = {}      % Stores elongated candidates of each line
5878         k_list = {}      % And all letters with kashida
5879         pos_inline = 0   % Not yet used
5880
5881         for n in node.traverse_id(GLYPH, line.head) do
5882             pos_inline = pos_inline + 1 % To find where it is. Not used.
5883
5884             % Elongated glyphs
5885             if elong_map then
5886                 local locale = node.get_attribute(n, LOCALE)
5887                 if elong_map[locale] and elong_map[locale][n.font] and
5888                     elong_map[locale][n.font][n.char] then
5889                     table.insert(elongs, {node = n, locale = locale} )
5890                     node.set_attribute(n.prev, KASHIDA, 0)
5891                 end
5892             end
5893
5894             % Tatwil. First create a list of nodes marked with kashida. The
5895             % rest of nodes can be ignored. The list of used weights is build
5896             % when transforms with the key kashida= are declared.
5897             if Babel.kashida_wts then
5898                 local k_wt = node.get_attribute(n, KASHIDA)
5899                 if k_wt > 0 then % todo. parameter for multi inserts
5900                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5901                 end
5902             end
5903
5904         end % of node.traverse_id
5905

```

```

5906 if #elongs == 0 and #k_list == 0 then goto next_line end
5907 full = line.width
5908 shift = line.shift
5909 goal = full * Babel.arabic.justify_factor % A bit crude
5910 width = node.dimensions(line.head) % The 'natural' width
5911
5912 % == Elongated ==
5913 % Original idea taken from 'chickenize'
5914 while (#elongs > 0 and width < goal) do
5915     subst_done = true
5916     local x = #elongs
5917     local curr = elongs[x].node
5918     local oldchar = curr.char
5919     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5920     width = node.dimensions(line.head) % Check if the line is too wide
5921     % Substitute back if the line would be too wide and break:
5922     if width > goal then
5923         curr.char = oldchar
5924         break
5925     end
5926     % If continue, pop the just substituted node from the list:
5927     table.remove(elongs, x)
5928 end
5929
5930 % == Tatwil ==
5931 % Traverse the kashida node list so many times as required, until
5932 % the line is filled. The first pass adds a tatweel after each
5933 % node with kashida in the line, the second pass adds another one,
5934 % and so on. In each pass, add first the kashida with the highest
5935 % weight, then with lower weight and so on.
5936 if #k_list == 0 then goto next_line end
5937
5938 width = node.dimensions(line.head) % The 'natural' width
5939 k_curr = #k_list % Traverse backwards, from the end
5940 wt_pos = 1
5941
5942 while width < goal do
5943     subst_done = true
5944     k_item = k_list[k_curr].node
5945     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5946         d = node.copy(k_item)
5947         d.char = 0x0640
5948         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5949         d.xoffset = 0
5950         line.head, new = node.insert_after(line.head, k_item, d)
5951         width_new = node.dimensions(line.head)
5952         if width > goal or width == width_new then
5953             node.remove(line.head, new) % Better compute before
5954             break
5955         end
5956         if Babel.fix_diacr then
5957             Babel.fix_diacr(k_item.next)
5958         end
5959         width = width_new
5960     end
5961     if k_curr == 1 then
5962         k_curr = #k_list
5963         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5964     else
5965         k_curr = k_curr - 1
5966     end
5967 end
5968

```

```

5969 % Limit the number of tatweel by removing them. Not very efficient,
5970 % but it does the job in a quite predictable way.
5971 if Babel.arabic.kashida_limit > -1 then
5972   cnt = 0
5973   for n in node.traverse_id(GLYPH, line.head) do
5974     if n.char == 0x0640 then
5975       cnt = cnt + 1
5976       if cnt > Babel.arabic.kashida_limit then
5977         node.remove(line.head, n)
5978       end
5979     else
5980       cnt = 0
5981     end
5982   end
5983 end
5984
5985 ::next_line::
5986
5987 % Must take into account marks and ins, see luatex manual.
5988 % Have to be executed only if there are changes. Investigate
5989 % what's going on exactly.
5990 if subst_done and not gc then
5991   d = node.hpack(line.head, full, 'exactly')
5992   d.shift = shift
5993   node.insert_before(head, line, d)
5994   node.remove(head, line)
5995 end
5996 end % if process line
5997 end
5998 }
5999 \endgroup
6000 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

## 10.9. Common stuff

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

```

6001 \def\bbl@scr@node@list{%
6002   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6003   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6004 \ifnum\bbl@bidimode=102 % bidi-r
6005   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6006 \fi
6007 \def\bbl@set@renderer{%
6008   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6009   \ifin@
6010     \let\bbl@unset@renderer\relax
6011   \else
6012     \bbl@exp{%
6013       \def\\bbl@unset@renderer{%
6014         \def<g__fontspec_default_fontopts_clist>{%
6015           \[g__fontspec_default_fontopts_clist]}}%
6016         \def<g__fontspec_default_fontopts_clist>{%
6017           Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
6018       \fi}
6019 <@Font selection@>

```

## 10.10 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a the function `Babel.locale_map`, which just traverse the node list to carry out the replacements. The table `loc_to_scr` stores the script range for each locale (whose id is the key), copied from this table (so that it can be modified on a locale basis); there is an intermediate table named `chr_to_loc` built on the fly for optimization, which maps a char to the locale. This locale is then used to get the `\language` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
6020 \directlua{% DL6
6021 Babel.script_blocks = {
6022   ['dflt'] = {},
6023   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6024             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6025   ['Armn'] = {{0x0530, 0x058F}},
6026   ['Beng'] = {{0x0980, 0x09FF}},
6027   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6028   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6029   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6030             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6031   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6032   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6033             {0xAB00, 0xAB2F}},
6034   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6035   % Don't follow strictly Unicode, which places some Coptic letters in
6036   % the 'Greek and Coptic' block
6037   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6038   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6039             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6040             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6041             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6042             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6043             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6044   ['Hebr'] = {{0x0590, 0x05FF},
6045             {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6046   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6047             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6048   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6049   ['Knda'] = {{0x0C80, 0x0CFF}},
6050   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6051             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6052             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6053   ['Lao'] = {{0x0E80, 0x0EFF}},
6054   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6055             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6056             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6057   ['Mahj'] = {{0x11150, 0x1117F}},
6058   ['Mlym'] = {{0x0D00, 0x0D7F}},
6059   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6060   ['Orya'] = {{0x0B00, 0x0B7F}},
6061   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6062   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6063   ['Taml'] = {{0x0B80, 0x0BFF}},
6064   ['Telu'] = {{0x0C00, 0x0C7F}},
6065   ['Tfng'] = {{0x2D30, 0x2D7F}},
6066   ['Thai'] = {{0x0E00, 0x0E7F}},
6067   ['Tibt'] = {{0x0F00, 0x0FFF}},
6068   ['Vaii'] = {{0xA500, 0xA63F}},
6069   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6070 }
6071
6072 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
```



```

6073 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6074 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6075
6076 function Babel.locale_map(head)
6077   if not Babel.locale_mapped then return head end
6078
6079   local LOCALE = Babel.attr_locale
6080   local GLYPH = node.id('glyph')
6081   local inmath = false
6082   local toloc_save
6083   for item in node.traverse(head) do
6084     local toloc
6085     if not inmath and item.id == GLYPH then
6086       % Optimization: build a table with the chars found
6087       if Babel.chr_to_loc[item.char] then
6088         toloc = Babel.chr_to_loc[item.char]
6089       else
6090         for lc, maps in pairs(Babel.loc_to_scr) do
6091           for _, rg in pairs(maps) do
6092             if item.char >= rg[1] and item.char <= rg[2] then
6093               Babel.chr_to_loc[item.char] = lc
6094               toloc = lc
6095               break
6096             end
6097           end
6098         end
6099         % Treat composite chars in a different fashion, because they
6100         % 'inherit' the previous locale.
6101         if (item.char >= 0x0300 and item.char <= 0x036F) or
6102            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6103            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6104           Babel.chr_to_loc[item.char] = -2000
6105           toloc = -2000
6106         end
6107         if not toloc then
6108           Babel.chr_to_loc[item.char] = -1000
6109         end
6110       end
6111       if toloc == -2000 then
6112         toloc = toloc_save
6113       elseif toloc == -1000 then
6114         toloc = nil
6115       end
6116       if toloc and Babel.locale_props[toloc] and
6117          Babel.locale_props[toloc].letters and
6118          tex.getcatcode(item.char) \string~= 11 then
6119         toloc = nil
6120       end
6121       if toloc and Babel.locale_props[toloc].script
6122          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6123          and Babel.locale_props[toloc].script ==
6124          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6125         toloc = nil
6126       end
6127       if toloc then
6128         if Babel.locale_props[toloc].lg then
6129           item.lang = Babel.locale_props[toloc].lg
6130           node.set_attribute(item, LOCALE, toloc)
6131         end
6132         if Babel.locale_props[toloc]['/'..item.font] then
6133           item.font = Babel.locale_props[toloc]['/'..item.font]
6134         end
6135       end

```

```

6136     toloc_save = toloc
6137     elseif not inmath and item.id == 7 then % Apply recursively
6138         item.replace = item.replace and Babel.locale_map(item.replace)
6139         item.pre      = item.pre and Babel.locale_map(item.pre)
6140         item.post     = item.post and Babel.locale_map(item.post)
6141     elseif item.id == node.id'math' then
6142         inmath = (item.subtype == 0)
6143     end
6144 end
6145 return head
6146 end
6147 }

```

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

```

6148 \newcommand\babelcharproperty[1]{%
6149   \count@=#1\relax
6150   \ifvmode
6151     \expandafter\bbl@chprop
6152   \else
6153     \bbl@error{charproperty-only-vertical}{}{}{}%
6154   \fi}
6155 \newcommand\bbl@chprop[3][\the\count@]{%
6156   \@tempcnta=#1\relax
6157   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6158   {\bbl@error{unknown-char-property}{}{#2}{}%
6159   {}%
6160   \loop
6161     \bbl@cs{chprop@#2}{#3}%
6162   \ifnum\count@<\@tempcnta
6163     \advance\count@\@ne
6164   \repeat}
6165 %
6166 \def\bbl@chprop@direction#1{%
6167   \directlua{
6168     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6169     Babel.characters[\the\count@]['d'] = '#1'
6170   }}
6171 \let\bbl@chprop@bc\bbl@chprop@direction
6172 %
6173 \def\bbl@chprop@mirror#1{%
6174   \directlua{
6175     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6176     Babel.characters[\the\count@]['m'] = '\number#1'
6177   }}
6178 \let\bbl@chprop@bmg\bbl@chprop@mirror
6179 %
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 %
6187 \def\bbl@chprop@locale#1{%
6188   \directlua{
6189     Babel.chr_to_loc = Babel.chr_to_loc or {}
6190     Babel.chr_to_loc[\the\count@] =
6191       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6192   }}

```

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.

```

6193 \directlua{% DL7

```

```

6194 Babel.nohyphenation = \the\l@nohyphenation
6195 }

```

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

```

6196 \begingroup
6197 \catcode\~ = 12
6198 \catcode\% = 12
6199 \catcode\& = 14
6200 \catcode\_| = 12
6201 \gdef\babelprehyphenation{%&%
6202   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{]}
6203 \gdef\babelposthyphenation{%&%
6204   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}{]}
6205 %
6206 \gdef\bbl@settransform#1[#2]#3#4#5{%&%
6207   \ifcase#1
6208     \bbl@activateprehyphen
6209   \or
6210     \bbl@activateposthyphen
6211   \fi
6212 \begingroup
6213   \def\babeltempa{\bbl@add@list\babeltempb}%&%
6214   \let\babeltempb\empty
6215   \def\bbl@tempa{#5}%&%
6216   \bbl@replace\bbl@tempa{,}{,}%&% TODO. Ugly trick to preserve {}
6217   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&%
6218     \bbl@ifsamestring{##1}{remove}%&%
6219     {\bbl@add@list\babeltempb{nil}}}%&%
6220   {\directlua{
6221     local rep = {[##1]=}
6222     local three_args = '%s*=%s*([%-d%.%a{ }|]+)%s+([%-d%.%a{ }|]+)%s+([%-d%.%a{ }|]+)'
6223     &% Numeric passes directly: kern, penalty...
6224     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6225     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6226     rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6227     rep = rep:gsub('(string)%s*=%s*([%^s,]*)', Babel.capture_func)
6228     rep = rep:gsub('node%s*=%s*([%a+)%s*([%a+])', Babel.capture_node)
6229     rep = rep:gsub(' (norule)' .. three_args,
6230       'norule = {' .. '%2, %3, %4' .. '}')
6231     if #1 == 0 or #1 == 2 then
6232       rep = rep:gsub(' (space)' .. three_args,
6233         'space = {' .. '%2, %3, %4' .. '}')
6234       rep = rep:gsub(' (spacefactor)' .. three_args,
6235         'spacefactor = {' .. '%2, %3, %4' .. '}')
6236       rep = rep:gsub(' (kashida)%s*=%s*([%^s,]*)', Babel.capture_kashida)
6237       &% Transform values
6238       rep, n = rep:gsub(' ({[%a%-%.]+}|([[%a%_%.]+)})',
6239         function(v,d)
6240           return string.format (
6241             '\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6242             v,
6243             load( 'return Babel.locale_props'..'
6244               '[\the\csname bbl@id@@#3\endcsname].' .. d)() )
6245           end )
6246       rep, n = rep:gsub(' ({[%a%-%.]+}|([%-d%.]+))',

```

```

6247         '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6248     end
6249     if #1 == 1 then
6250         rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6251         rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6252         rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6253     end
6254     tex.print([[string\babeltempa{[]] .. rep .. [[]]])
6255 }}&%
6256 \bbl@foreach\babeltempb{&%
6257     \bbl@forkv{##1}{&%
6258         \in@{,###1,},{,nil,step,data,remove,insert,string,no,pre,no,&%
6259             post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6260         \ifin@else
6261             \bbl@error{bad-transform-option}{###1}{}{&%
6262             \fi}}&%
6263 \let\bbl@kv@attribute\relax
6264 \let\bbl@kv@label\relax
6265 \let\bbl@kv@fonts\@empty
6266 \let\bbl@kv@prepend\relax
6267 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6268 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6269 \ifx\bbl@kv@attribute\relax
6270     \ifx\bbl@kv@label\relax\else
6271         \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6272         \bbl@replace\bbl@kv@fonts{ }{,}&%
6273         \edef\bbl@kv@attribute{\bbl@ATR@bbl@kv@label @#3@bbl@kv@fonts}&%
6274         \count@ \z@
6275         \def\bbl@elt##1##2##3{&%
6276             \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6277             {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6278             {\count@ \one}&%
6279             {\bbl@error{font-conflict-transforms}{}{}}}&%
6280             }}&%
6281         \bbl@transfont@list
6282         \ifnum\count@=\z@
6283             \bbl@exp{\global\bbl@add\bbl@transfont@list
6284                 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6285             \fi
6286             \bbl@ifunset{\bbl@kv@attribute}&%
6287             {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6288             {}&%
6289             \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6290         \fi
6291     \else
6292         \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6293     \fi
6294 \directlua{
6295     local lbkr = Babel.linebreaking.replacements[#1]
6296     local u = unicode.utf8
6297     local id, attr, label
6298     if #1 == 0 then
6299         id = \the\csname bbl@id@@#3\endcsname\space
6300     else
6301         id = \the\csname l@#3\endcsname\space
6302     end
6303     \ifx\bbl@kv@attribute\relax
6304         attr = -1
6305     \else
6306         attr = luatexbase.registernumber'\bbl@kv@attribute'
6307     \fi
6308     \ifx\bbl@kv@label\relax\else &% Same refs:
6309         label = [==[\bbl@kv@label]==]

```

```

6310 \fi
6311 &% Convert pattern:
6312 local patt = string.gsub([==[#4]==], '%s', '')
6313 if #1 == 0 then
6314     patt = string.gsub(patt, '|', ' ')
6315 end
6316 if not u.find(patt, '()', nil, true) then
6317     patt = '()' .. patt .. '()'
6318 end
6319 if #1 == 1 then
6320     patt = string.gsub(patt, '%(%)^', '^()')
6321     patt = string.gsub(patt, '%$(%)', '()$')
6322 end
6323 patt = u.gsub(patt, '{(.)}',
6324     function (n)
6325         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6326     end)
6327 patt = u.gsub(patt, '{(%x%x%x%+)}',
6328     function (n)
6329         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6330     end)
6331 lbkr[id] = lbkr[id] or {}
6332 table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6333 { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6334 }&%
6335 \endgroup}
6336 \endgroup
6337 %
6338 \let\bbl@transfont@list\empty
6339 \def\bbl@settransfont{%
6340 \global\let\bbl@settransfont\relax % Execute only once
6341 \gdef\bbl@transfont{%
6342 \def\bbl@elt####1####2####3{%
6343 \bbl@ifblank{####3}%
6344 {\count@tw@}% Do nothing if no fonts
6345 {\count@z@
6346 \bbl@vforeach{####3}{%
6347 \def\bbl@tempd{#####1}%
6348 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6349 \ifx\bbl@tempd\bbl@tempe
6350 \count@ne
6351 \else\ifx\bbl@tempd\bbl@transfam
6352 \count@ne
6353 \fi\fi}%
6354 \ifcase\count@
6355 \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6356 \or
6357 \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6358 \fi}}%
6359 \bbl@transfont@list}%
6360 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6361 \gdef\bbl@transfam{-unknown-}%
6362 \bbl@foreach\bbl@font@fams{%
6363 \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6364 \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6365 {\xdef\bbl@transfam{##1}}%
6366 {}}}
6367 %
6368 \DeclareRobustCommand\enablelocaletransform[1]{%
6369 \bbl@ifunset{\bbl@ATR@#1@\language @}%
6370 {\bbl@error{transform-not-available}{#1}{}}%
6371 {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6372 \DeclareRobustCommand\disablelocaletransform[1]{%

```

```

6373 \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6374 {\bbl@error{transform-not-available-b}{#1}{}}}%
6375 {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}

```

The following two macros load the Lua code for transforms, but only once. The only difference is in `add_after` and `add_before`.

```

6376 \def\bbl@activateposthyphen{%
6377 \let\bbl@activateposthyphen\relax
6378 \ifx\bbl@attr@hboxed\undefined
6379 \newattribute\bbl@attr@hboxed
6380 \fi
6381 \directlua{
6382   require('babel-transforms.lua')
6383   Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6384 }}
6385 \def\bbl@activateprehyphen{%
6386 \let\bbl@activateprehyphen\relax
6387 \ifx\bbl@attr@hboxed\undefined
6388 \newattribute\bbl@attr@hboxed
6389 \fi
6390 \directlua{
6391   require('babel-transforms.lua')
6392   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6393 }}
6394 \newcommand\SetTransformValue[3]{%
6395 \directlua{
6396   Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6397 }}

```

The code in `babel-transforms.lua` prints at some points the current string being transformed. This macro first make sure this file is loaded. Then, activates temporarily this feature and typeset inside a box the text in the argument.

```

6398 \newcommand\ShowBabelTransforms[1]{%
6399 \bbl@activateprehyphen
6400 \bbl@activateposthyphen
6401 \begin{group}
6402 \directlua{ Babel.show_transforms = true }%
6403 \setbox\z@\vbox{#1}%
6404 \directlua{ Babel.show_transforms = false }%
6405 \end{group}}

```

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.

```

6406 \newcommand\localeprehyphenation[1]{%
6407 \directlua{ Babel.string_prehyphenation([=#1]=], \the\localeid) }}

```

## 10.11.Bidi

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

```

6408 \def\bbl@activate@preotf{%
6409 \let\bbl@activate@preotf\relax % only once
6410 \directlua{
6411   function Babel.pre_otfload_v(head)
6412     if Babel.numbers and Babel.digits_mapped then
6413       head = Babel.numbers(head)
6414     end
6415     if Babel.bidi_enabled then
6416       head = Babel.bidi(head, false, dir)

```

```

6417     end
6418     return head
6419 end
6420 %
6421 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6422     if Babel.numbers and Babel.digits_mapped then
6423         head = Babel.numbers(head)
6424     end
6425     if Babel.bidi_enabled then
6426         head = Babel.bidi(head, false, dir)
6427     end
6428     return head
6429 end
6430 %
6431 luatexbase.add_to_callback('pre_linebreak_filter',
6432     Babel.pre_otfload_v,
6433     'Babel.pre_otfload_v',
6434     luatexbase.priority_in_callback('pre_linebreak_filter',
6435         'luaotfload.node_processor') or nil)
6436 %
6437 luatexbase.add_to_callback('hpack_filter',
6438     Babel.pre_otfload_h,
6439     'Babel.pre_otfload_h',
6440     luatexbase.priority_in_callback('hpack_filter',
6441         'luaotfload.node_processor') or nil)
6442 }}

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option `bidi=`. The hack for the PUA is no longer necessary with basic (24.8), but it's kept in `basic-r`.

```

6443 \breakafterdirmode=1
6444 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6445     \let\bbl@beforeforeign\leavevmode
6446     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6447     \RequirePackage{luatexbase}
6448     \bbl@activate@preotf
6449     \directlua{
6450         require('babel-data-bidi.lua')
6451         \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6452             require('babel-bidi-basic.lua')
6453         \or
6454             require('babel-bidi-basic-r.lua')
6455             table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6456             table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6457             table.insert(Babel.ranges, {0x10000, 0x10FFFD, 'on'})
6458         \fi}
6459     \newattribute\bbl@attr@dir
6460     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6461     \bbl@exp{\output{\bodydir\pagedir\the\output}}
6462 \fi
6463 %
6464 \chardef\bbl@thetextdir\z@
6465 \chardef\bbl@thepardir\z@
6466 \def\bbl@getluadir#1{%
6467     \directlua{
6468         if tex.#ldir == 'TLT' then
6469             tex.sprint('0')
6470         elseif tex.#ldir == 'TRT' then
6471             tex.sprint('1')
6472         else
6473             tex.sprint('0')
6474         end}}

```

```

6475 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6476 \ifcase#3\relax
6477 \ifcase\bbl@getluadir{#1}\relax\else
6478 #2 TLT\relax
6479 \fi
6480 \else
6481 \ifcase\bbl@getluadir{#1}\relax
6482 #2 TRT\relax
6483 \fi
6484 \fi}

```

\bbl@attr@dir stores the directions with a mask: ..00PPTT, with masks 0xC (PP is the par dir) and 0x3 (TT is the text dir).

```

6485 \def\bbl@thedir{0}
6486 \def\bbl@textdir#1{%
6487 \bbl@setluadir{text}\textdir{#1}%
6488 \chardef\bbl@thetextdir#1\relax
6489 \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6490 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6491 \def\bbl@pardir#1{% Used twice
6492 \bbl@setluadir{par}\pardir{#1}%
6493 \chardef\bbl@thepardir#1\relax}
6494 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6495 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6496 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6497 \ifnum\bbl@bidimode>\z@ % Any bidi=
6498 \def\bbl@insidemath{0}%
6499 \def\bbl@everymath{\def\bbl@insidemath{1}}
6500 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6501 \frozen@everymath\expandafter{%
6502 \expandafter\bbl@everymath\the\frozen@everymath}
6503 \frozen@everydisplay\expandafter{%
6504 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6505 \AtBeginDocument{
6506 \directlua{
6507 function Babel.math_box_dir(head)
6508 if not (token.get_macro('bbl@insidemath') == '0') then
6509 if Babel.hlist_has_bidi(head) then
6510 local d = node.new(node.id'dir')
6511 d.dir = '+TRT'
6512 node.insert_before(head, node.has_glyph(head), d)
6513 local inmath = false
6514 for item in node.traverse(head) do
6515 if item.id == 11 then
6516 inmath = (item.subtype == 0)
6517 elseif not inmath then
6518 node.set_attribute(item,
6519 Babel.attr_dir, token.get_macro('bbl@thedir'))
6520 end
6521 end
6522 end
6523 end
6524 return head
6525 end
6526 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6527 "Babel.math_box_dir", 0)
6528 if Babel.unset_atdir then
6529 luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6530 "Babel.unset_atdir")
6531 luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6532 "Babel.unset_atdir")

```



```

6533     end
6534 }}%
6535 \fi

Experimental. Tentative name.

6536 \DeclareRobustCommand\localebox[1]{%
6537   {\def\bbl@insidemath{0}%
6538     \mbox{\foreignlanguage{\language}\{#1\}}}}

```

## 10.12 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidibasic`, 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.

```

6539 \bbl@trace{Redefinitions for bidi layout}
6540 %
6541 <<(*More package options)>> ≡
6542 \chardef\bbl@eqnpos\z@
6543 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6544 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6545 <</More package options>>
6546 %
6547 \ifnum\bbl@bidimode>\z@ % Any bidi=
6548   \matheqdirmode\@ne % A luatex primitive
6549   \let\bbl@eqnodir\relax
6550   \def\bbl@eqdel{()}
6551   \def\bbl@eqnum{%
6552     {\normalfont\normalcolor
6553       \expandafter\@firstoftwo\bbl@eqdel
6554       \theequation
6555       \expandafter\@secondoftwo\bbl@eqdel}}
6556   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6557   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6558   \def\bbl@eqno@flip#1{%
6559     \ifdim\predisplaysize=-\maxdimen
6560       \eqno
6561       \hb@xt@.01pt{%
6562         \hb@xt@\displaywidth{\hss#1\glet\bbl@upset\@currentlabel}}\hss}%
6563     \else
6564       \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6565     \fi
6566     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6567   \def\bbl@leqno@flip#1{%
6568     \ifdim\predisplaysize=-\maxdimen
6569       \leqno
6570       \hb@xt@.01pt{%
6571         \hss\hb@xt@\displaywidth{\#1\glet\bbl@upset\@currentlabel}}\hss}%

```

```

6572 \else
6573 \eqno\hbox{#1\glet\bb@upset\@currentlabel}%
6574 \fi
6575 \bb@exp{\def\\@currentlabel{\bb@upset}}}%
6576 %
6577 \AtBeginDocument{%
6578 \ifx\bb@noamsmath\relax\else
6579 \ifx\maketag@@@ \@undefined % Normal equation, eqnarray
6580 \AddToHook{env/equation/begin}{%
6581 \ifnum\bb@thetextdir>\z@
6582 \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6583 \let\@eqnnum\bb@eqnum
6584 \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6585 \chardef\bb@thetextdir\z@
6586 \bb@add\normalfont{\bb@eqnodir}%
6587 \ifcase\bb@eqnpos
6588 \let\bb@puteqno\bb@eqno@flip
6589 \or
6590 \let\bb@puteqno\bb@leqno@flip
6591 \fi
6592 \fi}%
6593 \ifnum\bb@eqnpos=\tw@ \else
6594 \def\endequation{\bb@puteqno{\@eqnnum}$$\@ignoretrue}%
6595 \fi
6596 \AddToHook{env/eqnarray/begin}{%
6597 \ifnum\bb@thetextdir>\z@
6598 \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6599 \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6600 \chardef\bb@thetextdir\z@
6601 \bb@add\normalfont{\bb@eqnodir}%
6602 \ifnum\bb@eqnpos=\@ne
6603 \def\@eqnnum{%
6604 \setbox\z@\hbox{\bb@eqnum}%
6605 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6606 \else
6607 \let\@eqnnum\bb@eqnum
6608 \fi
6609 \fi}
6610 % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6611 \expandafter\bb@sreplace\csname \endcsname{$$}{\eqno\kern.001pt$}$%
6612 \else % amstex
6613 \bb@exp{% Hack to hide maybe undefined conditionals:
6614 \chardef\bb@eqnpos=0%
6615 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6616 \ifnum\bb@eqnpos=\@ne
6617 \let\bb@ams@lap\hbox
6618 \else
6619 \let\bb@ams@lap\llap
6620 \fi
6621 \ExplSyntaxOn % Required by \bb@sreplace with \intertext@
6622 \bb@sreplace\intertext@\normalbaselines%
6623 {\normalbaselines
6624 \ifx\bb@eqnodir\relax\else\bb@pardir\@ne\bb@eqnodir\fi}%
6625 \ExplSyntaxOff
6626 \def\bb@ams@tagbox#1#2{#1{\bb@eqnodir#2}}% #1=hbox|@lap|flip
6627 \ifx\bb@ams@lap\hbox % leqno
6628 \def\bb@ams@flip#1{%
6629 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6630 \else % eqno
6631 \def\bb@ams@flip#1{%
6632 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}%
6633 \fi
6634 \def\bb@ams@preset#1{%

```

```

6635 \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6636 \ifnum\bb@thetextdir>\z@
6637 \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6638 \bb@replace\textdef@{\hbox}{\bb@ams@tagbox\hbox}%
6639 \bb@replace\maketag@@@{\hbox}{\bb@ams@tagbox#1}%
6640 \fi}%
6641 \ifnum\bb@eqnpos=\tw@else
6642 \def\bb@ams@equation{%
6643 \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6644 \ifnum\bb@thetextdir>\z@
6645 \edef\bb@eqnodir{\noexpand\bb@textdir{\the\bb@thetextdir}}%
6646 \chardef\bb@thetextdir\z@
6647 \bb@add\normalfont{\bb@eqnodir}%
6648 \ifcase\bb@eqnpos
6649 \def\veqno##1##2{\bb@eqno@flip{##1##2}}%
6650 \or
6651 \def\veqno##1##2{\bb@leqno@flip{##1##2}}%
6652 \fi
6653 \fi}%
6654 \AddToHook{env/equation/begin}{\bb@ams@equation}%
6655 \AddToHook{env/equation*/begin}{\bb@ams@equation}%
6656 \fi
6657 \AddToHook{env/cases/begin}{\bb@ams@preset\bb@ams@lap}%
6658 \AddToHook{env/multline/begin}{\bb@ams@preset\hbox}%
6659 \AddToHook{env/gather/begin}{\bb@ams@preset\bb@ams@lap}%
6660 \AddToHook{env/gather*/begin}{\bb@ams@preset\bb@ams@lap}%
6661 \AddToHook{env/align/begin}{\bb@ams@preset\bb@ams@lap}%
6662 \AddToHook{env/align*/begin}{\bb@ams@preset\bb@ams@lap}%
6663 \AddToHook{env/alignat/begin}{\bb@ams@preset\bb@ams@lap}%
6664 \AddToHook{env/alignat*/begin}{\bb@ams@preset\bb@ams@lap}%
6665 \AddToHook{env/eqnalign/begin}{\bb@ams@preset\hbox}%
6666 % Hackish, for proper alignment. Don't ask me why it works!
6667 \bb@exp{% Avoid a 'visible' conditional
6668 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}\<fi>}%
6669 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{}\<fi>}}%
6670 \AddToHook{env/flalign/begin}{\bb@ams@preset\hbox}%
6671 \AddToHook{env/split/before}{%
6672 \def\bb@mathboxdir{\def\bb@insidemath{1}}%
6673 \ifnum\bb@thetextdir>\z@
6674 \bb@ifsamestring\@currentvir{equation}%
6675 {\ifx\bb@ams@lap\hbox % leqno
6676 \def\bb@ams@flip#1{%
6677 \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6678 \else
6679 \def\bb@ams@flip#1{%
6680 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6681 \fi}%
6682 {}%
6683 \fi}%
6684 \fi\fi}
6685 \fi

```

Declarations specific to lua, called by \babelprovide.

```

6686 \def\bb@provide@extra#1{%
6687 % == onchar ==
6688 \ifx\bb@KVP@onchar\@nnil\else
6689 \bb@luahyphenate
6690 \bb@exp{%
6691 \\\AddToHook{env/document/before}{\select@language{#1}}}%
6692 \directlua{
6693 if Babel.locale_mapped == nil then
6694 Babel.locale_mapped = true
6695 Babel.linebreaking.add_before(Babel.locale_map, 1)

```

```

6696     Babel.loc_to_scr = {}
6697     Babel.chr_to_loc = Babel.chr_to_loc or {}
6698 end
6699 Babel.locale_props[\the\localeid].letters = false
6700 }%
6701 \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6702 \ifin@
6703   \directlua{
6704     Babel.locale_props[\the\localeid].letters = true
6705   }%
6706 \fi
6707 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6708 \ifin@
6709   \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
6710     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}}%
6711   \fi
6712   \bbl@exp{\bbl@add\bbl@starthyphens
6713     {\bbl@patterns@lua{\language}}}%
6714   \directlua{
6715     if Babel.script_blocks['\bbl@cl{sbc}'] then
6716       Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6717       Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language}\space
6718     end
6719   }%
6720 \fi
6721 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6722 \ifin@
6723   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
6724   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
6725   \directlua{
6726     if Babel.script_blocks['\bbl@cl{sbc}'] then
6727       Babel.loc_to_scr[\the\localeid] =
6728         Babel.script_blocks['\bbl@cl{sbc}']
6729     end}%
6730   \ifx\bbl@mapselect\undefined
6731     \AtBeginDocument{%
6732       \bbl@patchfont{\bbl@mapselect}}%
6733     {\selectfont}}%
6734   \def\bbl@mapselect{%
6735     \let\bbl@mapselect\relax
6736     \edef\bbl@prefontid{\fontid\font}}%
6737   \def\bbl@mapdir##1{%
6738     \begingroup
6739       \setbox\z@\hbox{% Force text mode
6740         \def\language{##1}%
6741         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6742         \bbl@switchfont
6743         \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6744           \directlua{
6745             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6746               ['\bbl@prefontid'] = \fontid\font\space}%
6747           \fi}%
6748         \endgroup}%
6749   \fi
6750   \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
6751 \fi
6752 \fi
6753 % == mapfont ==
6754 % For bidi texts, to switch the font based on direction. Deprecated
6755 \ifx\bbl@KVP@mapfont\@nnil\else
6756   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{%
6757     {\bbl@error{unknown-mapfont}}}%
6758   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%

```

```

6759 \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
6760 \ifx\bbl@mapselect\undefined
6761 \AtBeginDocument{%
6762 \bbl@patchfont{\bbl@mapselect}%
6763 {\selectfont}}%
6764 \def\bbl@mapselect{%
6765 \let\bbl@mapselect\relax
6766 \edef\bbl@prefontid{\fontid\font}}%
6767 \def\bbl@mapdir##1{%
6768 {\def\languagename{##1}%
6769 \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6770 \bbl@switchfont
6771 \directlua{Babel.fontmap
6772 [\the\csname bbl@wdir@##1\endcsname]%
6773 [\bbl@prefontid]=\fontid\font}}}%
6774 \fi
6775 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
6776 \fi
6777 % == Line breaking: CJK quotes ==
6778 \ifcase\bbl@engine\or
6779 \bbl@xin@{/c}{\bbl@ccl{\lnbrk}}%
6780 \ifin@
6781 \bbl@ifunset{\bbl@quote@\languagename}{}%
6782 {\directlua{
6783 Babel.locale_props[\the\localeid].cjk_quotes = {}
6784 local cs = 'op'
6785 for c in string.utfvalues(
6786 [[\csname bbl@quote@\languagename\endcsname]]) do
6787 if Babel.cjk_characters[c].c == 'qu' then
6788 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6789 end
6790 cs = ( cs == 'op') and 'cl' or 'op'
6791 end
6792 }}%
6793 \fi
6794 \fi
6795 % == Counters: mapdigits ==
6796 % Native digits
6797 \ifx\bbl@KVP@mapdigits\@nnil\else
6798 \bbl@ifunset{\bbl@dgnat@\languagename}{}%
6799 {\RequirePackage{luatexbase}%
6800 \bbl@activate@preotf
6801 \directlua{
6802 Babel.digits_mapped = true
6803 Babel.digits = Babel.digits or {}
6804 Babel.digits[\the\localeid] =
6805 table.pack(string.utfvalue('\bbl@ccl{dgnat}'))
6806 if not Babel.numbers then
6807 function Babel.numbers(head)
6808 local LOCALE = Babel.attr_locale
6809 local GLYPH = node.id'glyph'
6810 local inmath = false
6811 for item in node.traverse(head) do
6812 if not inmath and item.id == GLYPH then
6813 local temp = node.get_attribute(item, LOCALE)
6814 if Babel.digits[temp] then
6815 local chr = item.char
6816 if chr > 47 and chr < 58 then
6817 item.char = Babel.digits[temp][chr-47]
6818 end
6819 end
6820 elseif item.id == node.id'math' then
6821 inmath = (item.subtype == 0)

```

```

6822         end
6823     end
6824     return head
6825 end
6826 end
6827 }}%
6828 \fi
6829 % == transforms ==
6830 \ifx\bbk@KVP@transforms\@nnil\else
6831   \def\bbk@elt##1##2##3{%
6832     \in@{$transforms.}{$##1}%
6833     \ifin@
6834       \def\bbk@tempa{##1}%
6835       \bbk@replace\bbk@tempa{transforms.}{}%
6836       \bbk@carg\bbk@transforms{babel\bbk@tempa}{##2}{##3}%
6837     \fi}%
6838 \bbk@exp{%
6839   \\bbk@ifblank{\bbk@cl{dgnat}}}%
6840   {\let\\bbk@tempa\relax}%
6841   {\def\\bbk@tempa{%
6842     \\bbk@elt{transforms.prehyphenation}%
6843     {digits.native.1.0}{([0-9])}%
6844     \\bbk@elt{transforms.prehyphenation}%
6845     {digits.native.1.1}{string={1string|0123456789string|\bbk@cl{dgnat}}}}}%
6846 \ifx\bbk@tempa\relax\else
6847   \toks@ \expandafter\expandafter\expandafter{%
6848     \csname \bbk@inidata@\language\endcsname}%
6849     \bbk@csarg\edef\inidata@\language}%
6850   \unexpanded\expandafter{\bbk@tempa}%
6851   \the\toks@}%
6852 \fi
6853 \csname \bbk@inidata@\language\endcsname
6854 \bbk@release@transforms\relax % \relax closes the last item.
6855 \fi}

```

Start tabular here:

```

6856 \def\localerestoredirs{%
6857   \ifcase\bbk@thetextdir
6858     \ifnum\textdirection=\z@ \else \textdir TLT \fi
6859   \else
6860     \ifnum\textdirection=\@ne \else \textdir TRT \fi
6861   \fi
6862   \ifcase\bbk@thepardir
6863     \ifnum\pardirection=\z@ \else \pardir TLT\bodydir TLT \fi
6864   \else
6865     \ifnum\pardirection=\@ne \else \pardir TRT\bodydir TRT \fi
6866   \fi}
6867 %
6868 \IfBabelLayout{tabular}%
6869   {\chardef\bbk@tabular@mode\tw@}% All RTL
6870   {\IfBabelLayout{notabular}%
6871     {\chardef\bbk@tabular@mode\z@}%
6872     {\chardef\bbk@tabular@mode\@ne}}% Mixed, with LTR cols
6873 %
6874 \ifnum\bbk@bidimode>\@ne % Any lua bidi= except default=1
6875   % Redefine: vrules mess up dirs.
6876   \def\@arstrut{\relax\copy\@arstrutbox}%
6877   \ifcase\bbk@tabular@mode\or % 1 = Mixed - default
6878     \let\bbk@parabefore\relax
6879     \AddToHook{para/before}{\bbk@parabefore}
6880     \AtBeginDocument{%
6881       \bbk@replace\@tabular{$}{$%
6882         \def\bbk@insidemath{0}%

```

```

6883 \def\bbl@parabefore{\localerestoredirs}}%
6884 \ifnum\bbl@tabular@mode=\@ne
6885 \bbl@ifunset{@tabclassz}{}%
6886 \bbl@exp{% Hide conditionals
6887 \\\bbl@sreplace\\\@tabclassz
6888 {\<ifcase>\\\@chnum}%
6889 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6890 \@ifpackageloaded{colortbl}%
6891 {\bbl@sreplace\@classz
6892 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6893 {\@ifpackageloaded{array}%
6894 {\bbl@exp{% Hide conditionals
6895 \\\bbl@sreplace\\\@classz
6896 {\<ifcase>\\\@chnum}%
6897 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6898 \\\bbl@sreplace\\\@classz
6899 {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
6900 {}}%
6901 \fi}%
6902 \or % 2 = All RTL - tabular
6903 \let\bbl@parabefore\relax
6904 \AddToHook{para/before}{\bbl@parabefore}%
6905 \AtBeginDocument{%
6906 \@ifpackageloaded{colortbl}%
6907 {\bbl@replace\@tabular{$}{$}%
6908 \def\bbl@insidemath{0}%
6909 \def\bbl@parabefore{\localerestoredirs}}%
6910 \bbl@sreplace\@classz
6911 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6912 {}}%
6913 \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.

```

6914 \AtBeginDocument{%
6915 \@ifpackageloaded{multicol}%
6916 {\toks@ \expandafter{\multi@column@out}%
6917 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6918 {}%
6919 \@ifpackageloaded{paracol}%
6920 {\edef\pcol@output{%
6921 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6922 {}}%
6923 \fi

```

Finish here if there in no layout.

```

6924 \ifx\bbl@opt@layout\@nnil\endinput\fi

```

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. Used in `tabular`, `\underline` and `\LaTeX`. Also, `\hangindent` does not honour direction changes by default, so we need to redefine `\@hangfrom`.

```

6925 \ifnum\bbl@bidimode>\z@ % Any bidi=
6926 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6927 \bbl@exp{%
6928 \mathdir\the\bodydir
6929 #1% Once entered in math, set boxes to restore values
6930 \def\\\bbl@insidemath{0}%
6931 \<ifmmode>%
6932 \everyvbox{%
6933 \the\everyvbox
6934 \bodydir\the\bodydir

```

```

6935         \mathdir\the\mathdir
6936         \everyhbox{\the\everyhbox}%
6937         \everyvbox{\the\everyvbox}}}%
6938     \everyhbox{%
6939         \the\everyhbox
6940         \bodydir\the\bodydir
6941         \mathdir\the\mathdir
6942         \everyhbox{\the\everyhbox}%
6943         \everyvbox{\the\everyvbox}}}%
6944     \<fi>}}%
6945 \IfBabelLayout{nopars}
6946 {}
6947 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
6948 \IfBabelLayout{pars}
6949 {\def\@hangfrom#1{%
6950     \setbox\@tempboxa\hbox{{#1}}%
6951     \hangindent\wd\@tempboxa
6952     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6953         \shapemode\@ne
6954     \fi
6955     \noindent\box\@tempboxa}}
6956 {}
6957 \fi
6958 %
6959 \IfBabelLayout{tabular}
6960 {\let\bbl@OL@tabular\@tabular
6961  \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6962  \let\bbl@NL@tabular\@tabular
6963  \AtBeginDocument{%
6964      \ifx\bbl@NL@tabular\@tabular\else
6965          \bbl@exp{\in{\bbl@nextfake}{\@tabular}}}%
6966      \ifin\else
6967          \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6968      \fi
6969      \let\bbl@NL@tabular\@tabular
6970      \fi}}
6971 {}
6972 %
6973 \IfBabelLayout{lists}
6974 {\let\bbl@OL@list\list
6975  \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6976  \let\bbl@NL@list\list
6977  \def\bbl@listparshape#1#2#3{%
6978      \parshape #1 #2 #3 %
6979      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6980          \shapemode\tw@
6981      \fi}}
6982 {}
6983 %
6984 \IfBabelLayout{graphics}
6985 {\let\bbl@pictresetdir\relax
6986  \def\bbl@pictsetdir#1{%
6987      \ifcase\bbl@thetextdir
6988          \let\bbl@pictresetdir\relax
6989      \else
6990          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6991              \or\textdir TLT
6992              \else\bodydir TLT \textdir TLT
6993          \fi
6994          % \ (text|par)dir required in pgf:
6995          \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6996      \fi}%
6997  \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%

```



```

6998 \directlua{
6999     Babel.get_picture_dir = true
7000     Babel.picture_has_bidi = 0
7001     %
7002     function Babel.picture_dir (head)
7003         if not Babel.get_picture_dir then return head end
7004         if Babel.hlist_has_bidi(head) then
7005             Babel.picture_has_bidi = 1
7006         end
7007         return head
7008     end
7009     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7010         "Babel.picture_dir")
7011 }%
7012 \AtBeginDocument{%
7013     \def\LS@rot{%
7014         \setbox\@outputbox\vbox{%
7015             \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
7016     \long\def\put(#1,#2)#3{%
7017         \@killglove
7018         % Try:
7019         \ifx\bbp@pictresetdir\relax
7020             \def\bbp@tempc{0}%
7021         \else
7022             \directlua{
7023                 Babel.get_picture_dir = true
7024                 Babel.picture_has_bidi = 0
7025             }%
7026             \setbox\z@\hb@xt@\z@{%
7027                 \@defaultunitsset\@tempdimc{#1}\unitlength
7028                 \kern\@tempdimc
7029                 #3\hss}%
7030             \edef\bbp@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7031         \fi
7032         % Do:
7033         \@defaultunitsset\@tempdimc{#2}\unitlength
7034         \raise\@tempdimc\hb@xt@\z@{%
7035             \@defaultunitsset\@tempdimc{#1}\unitlength
7036             \kern\@tempdimc
7037             {\ifnum\bbp@tempc>\z@\bbp@pictresetdir\fi#3}\hss}%
7038         \ignorespaces}%
7039     \MakeRobust\put}%
7040 \AtBeginDocument
7041 {\AddToHook{cmd/diagbox@pict/before}{\let\bbp@pictsetdir\@gobble}%
7042 \ifx\pgfpicture\undefined\else
7043     \AddToHook{env/pgfpicture/begin}{\bbp@pictsetdir\@ne}%
7044     \bbp@add\pgfinterruptpicture{\bbp@pictresetdir}%
7045     \bbp@add\pgfsys@beginpicture{\bbp@pictsetdir\z@}%
7046 \fi
7047 \ifx\tikzpicture\undefined\else
7048     \AddToHook{env/tikzpicture/begin}{\bbp@pictsetdir\tw}%
7049     \bbp@add\tikz@atbegin@node{\bbp@pictresetdir}%
7050     \bbp@sreplace\tikz{\begingroup}{\begingroup\bbp@pictsetdir\tw}%
7051     \bbp@sreplace\tikzpicture{\begingroup}{\begingroup\bbp@pictsetdir\tw}%
7052 \fi
7053 \ifx\tcolorbox\undefined\else
7054     \def\tcb@drawing@env@begin{%
7055         \csname tcb@before@tcb@split@state\endcsname
7056         \bbp@pictsetdir\tw@
7057         \begin{\kvtcb@graphenv}%
7058         \tcb@bbdraw
7059         \tcb@apply@graph@patches}%
7060     \def\tcb@drawing@env@end{%

```

```

7061      \end{\kv tcb@graphenv}%
7062      \bbl@pictresetdir
7063      \csname tcb@after@\tcb@split@state\endcsname}%
7064      \fi
7065    }}
7066  {}

```

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`.

```

7067 \IfBabelLayout{counters*}%
7068   {\bbl@add\bbl@opt@layout{.counters.}%
7069    \directlua{
7070      luatexbase.add_to_callback("process_output_buffer",
7071        Babel.discard_sublr , "Babel.discard_sublr") }%
7072   }{}
7073 \IfBabelLayout{counters}%
7074   {\let\bbl@0L@@textsuperscript\@textsuperscript
7075    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7076    \let\bbl@latinarabic=\@arabic
7077    \let\bbl@0L@@arabic\@arabic
7078    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7079    \ifpackagewith{babel}{bidi=default}%
7080      {\let\bbl@asciroman=\@roman
7081       \let\bbl@0L@@roman\@roman
7082       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7083       \let\bbl@asciiRoman=\@Roman
7084       \let\bbl@0L@@roman\@Roman
7085       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7086       \let\bbl@0L@labelenumii\labelenumii
7087       \def\labelenumii{}\theenumii}%
7088       \let\bbl@0L@p@enumiii\p@enumiii
7089       \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}

```

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

```

7090 \IfBabelLayout{extras}%
7091   {\bbl@ncarg\let\bbl@0L@underline{underline }%
7092    \bbl@carg\bbl@sreplace{underline }%
7093    {\$@@underline}{\bgroup\bbl@nextfake$\@underline}%
7094    \bbl@carg\bbl@sreplace{underline }%
7095    {\m@th$}{\m@th$\egroup}%
7096    \let\bbl@0L@LaTeXe\LaTeXe
7097    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7098      \if b\expandafter\@car\@series\@nil\boldmath\fi
7099      \babelsublr{%
7100        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
7101   {}
7102 \end{luatex}

```

## 10.13 Lua: transforms

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

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

```

7103 (*transforms)
7104 Babel.linebreaking.replacements = {}
7105 Babel.linebreaking.replacements[0] = {} -- pre
7106 Babel.linebreaking.replacements[1] = {} -- post
7107
7108 function Babel.tovalue(v)
7109   if type(v) == 'table' then
7110     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7111   else
7112     return v
7113   end
7114 end
7115
7116 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7117
7118 function Babel.set_hboxed(head, gc)
7119   for item in node.traverse(head) do
7120     node.set_attribute(item, Babel.attr_hboxed, 1)
7121   end
7122   return head
7123 end
7124
7125 Babel.fetch_subtext = {}
7126
7127 Babel.ignore_pre_char = function(node)
7128   return (node.lang == Babel.nohyphenation)
7129 end
7130
7131 Babel.show_transforms = false
7132
7133 -- Merging both functions doesn't seem feasible, because there are too
7134 -- many differences.
7135 Babel.fetch_subtext[0] = function(head)
7136   local word_string = ''
7137   local word_nodes = {}
7138   local lang
7139   local item = head
7140   local inmath = false
7141
7142   while item do
7143
7144     if item.id == 11 then
7145       inmath = (item.subtype == 0)
7146     end
7147
7148     if inmath then
7149       -- pass
7150
7151     elseif item.id == 29 then
7152       local locale = node.get_attribute(item, Babel.attr_locale)
7153
7154       if lang == locale or lang == nil then
7155         lang = lang or locale
7156         if Babel.ignore_pre_char(item) then
7157           word_string = word_string .. Babel.us_char
7158         else
7159           if node.has_attribute(item, Babel.attr_hboxed) then
7160             word_string = word_string .. Babel.us_char
7161           else
7162             word_string = word_string .. unicode.utf8.char(item.char)
7163           end
7164         end
7165         word_nodes[#word_nodes+1] = item

```

```

7166     else
7167         break
7168     end
7169
7170 elseif item.id == 12 and item.subtype == 13 then
7171     if node.has_attribute(item, Babel.attr_hboxed) then
7172         word_string = word_string .. Babel.us_char
7173     else
7174         word_string = word_string .. ' '
7175     end
7176     word_nodes[#word_nodes+1] = item
7177
7178     -- Ignore leading unrecognized nodes, too.
7179 elseif word_string ~= '' then
7180     word_string = word_string .. Babel.us_char
7181     word_nodes[#word_nodes+1] = item -- Will be ignored
7182 end
7183
7184 item = item.next
7185 end
7186
7187 -- Here and above we remove some trailing chars but not the
7188 -- corresponding nodes. But they aren't accessed.
7189 if word_string:sub(-1) == ' ' then
7190     word_string = word_string:sub(1,-2)
7191 end
7192 if Babel.show_transforms then texio.write_nl(word_string) end
7193 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7194 return word_string, word_nodes, item, lang
7195 end
7196
7197 Babel.fetch_subtext[1] = function(head)
7198     local word_string = ''
7199     local word_nodes = {}
7200     local lang
7201     local item = head
7202     local inmath = false
7203
7204     while item do
7205
7206         if item.id == 11 then
7207             inmath = (item.subtype == 0)
7208         end
7209
7210         if inmath then
7211             -- pass
7212         end
7213
7214         elseif item.id == 29 then
7215             if item.lang == lang or lang == nil then
7216                 lang = lang or item.lang
7217                 if node.has_attribute(item, Babel.attr_hboxed) then
7218                     word_string = word_string .. Babel.us_char
7219                 elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7220                     word_string = word_string .. Babel.us_char
7221                 else
7222                     word_string = word_string .. unicode.utf8.char(item.char)
7223                 end
7224                 word_nodes[#word_nodes+1] = item
7225             else
7226                 break
7227             end
7228
7229         elseif item.id == 7 and item.subtype == 2 then

```

```

7229     if node.has_attribute(item, Babel.attr_hboxed) then
7230         word_string = word_string .. Babel.us_char
7231     else
7232         word_string = word_string .. '='
7233     end
7234     word_nodes[#word_nodes+1] = item
7235
7236 elseif item.id == 7 and item.subtype == 3 then
7237     if node.has_attribute(item, Babel.attr_hboxed) then
7238         word_string = word_string .. Babel.us_char
7239     else
7240         word_string = word_string .. '|'
7241     end
7242     word_nodes[#word_nodes+1] = item
7243
7244     -- (1) Go to next word if nothing was found, and (2) implicitly
7245     -- remove leading USs.
7246 elseif word_string == '' then
7247     -- pass
7248
7249     -- This is the responsible for splitting by words.
7250 elseif (item.id == 12 and item.subtype == 13) then
7251     break
7252
7253 else
7254     word_string = word_string .. Babel.us_char
7255     word_nodes[#word_nodes+1] = item -- Will be ignored
7256 end
7257
7258 item = item.next
7259 end
7260 if Babel.show_transforms then texio.write_nl(word_string) end
7261 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7262 return word_string, word_nodes, item, lang
7263 end
7264
7265 function Babel.pre_hyphenate_replace(head)
7266     Babel.hyphenate_replace(head, 0)
7267 end
7268
7269 function Babel.post_hyphenate_replace(head)
7270     Babel.hyphenate_replace(head, 1)
7271 end
7272
7273 Babel.us_char = string.char(31)
7274
7275 function Babel.hyphenate_replace(head, mode)
7276     local u = unicode.utf8
7277     local lbkr = Babel.linebreaking.replacements[mode]
7278     local tovalue = Babel.tovalue
7279
7280     local word_head = head
7281
7282     if Babel.show_transforms then
7283         texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7284     end
7285
7286     while true do -- for each subtext block
7287
7288         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7289
7290         if Babel.debug then
7291             print()

```

```

7292     print((mode == 0) and '@@@<' or '@@@>', w)
7293 end
7294
7295 if nw == nil and w == '' then break end
7296
7297 if not lang then goto next end
7298 if not lbkr[lang] then goto next end
7299
7300 -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7301 -- loops are nested.
7302 for k=1, #lbkr[lang] do
7303     local p = lbkr[lang][k].pattern
7304     local r = lbkr[lang][k].replace
7305     local attr = lbkr[lang][k].attr or -1
7306
7307     if Babel.debug then
7308         print('*****', p, mode)
7309     end
7310
7311     -- This variable is set in some cases below to the first *byte*
7312     -- after the match, either as found by u.match (faster) or the
7313     -- computed position based on sc if w has changed.
7314     local last_match = 0
7315     local step = 0
7316
7317     -- For every match.
7318     while true do
7319         if Babel.debug then
7320             print('====')
7321         end
7322         local new -- used when inserting and removing nodes
7323         local dummy_node -- used by after
7324
7325         local matches = { u.match(w, p, last_match) }
7326
7327         if #matches < 2 then break end
7328
7329         -- Get and remove empty captures (with ()'s, which return a
7330         -- number with the position), and keep actual captures
7331         -- (from (...)), if any, in matches.
7332         local first = table.remove(matches, 1)
7333         local last = table.remove(matches, #matches)
7334         -- Non re-fetched substrings may contain \31, which separates
7335         -- subsubstrings.
7336         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7337
7338         local save_last = last -- with A()BC()D, points to D
7339
7340         -- Fix offsets, from bytes to unicode. Explained above.
7341         first = u.len(w:sub(1, first-1)) + 1
7342         last = u.len(w:sub(1, last-1)) -- now last points to C
7343
7344         -- This loop stores in a small table the nodes
7345         -- corresponding to the pattern. Used by 'data' to provide a
7346         -- predictable behavior with 'insert' (w_nodes is modified on
7347         -- the fly), and also access to 'remove'd nodes.
7348         local sc = first-1 -- Used below, too
7349         local data_nodes = {}
7350
7351         local enabled = true
7352         for q = 1, last-first+1 do
7353             data_nodes[q] = w_nodes[sc+q]
7354             if enabled

```

```

7355         and attr > -1
7356         and not node.has_attribute(data_nodes[q], attr)
7357     then
7358         enabled = false
7359     end
7360 end
7361
7362 -- This loop traverses the matched substring and takes the
7363 -- corresponding action stored in the replacement list.
7364 -- sc = the position in substr nodes / string
7365 -- rc = the replacement table index
7366 local rc = 0
7367
7368 ----- TODO. dummy_node?
7369 while rc < last-first+1 or dummy_node do -- for each replacement
7370     if Babel.debug then
7371         print('.....', rc + 1)
7372     end
7373     sc = sc + 1
7374     rc = rc + 1
7375
7376     if Babel.debug then
7377         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7378         local ss = ''
7379         for itt in node.traverse(head) do
7380             if itt.id == 29 then
7381                 ss = ss .. unicode.utf8.char(itt.char)
7382             else
7383                 ss = ss .. '{' .. itt.id .. '}'
7384             end
7385         end
7386         print('*****', ss)
7387     end
7388
7389     local crep = r[rc]
7390     local item = w_nodes[sc]
7391     local item_base = item
7392     local placeholder = Babel.us_char
7393     local d
7394
7395     if crep and crep.data then
7396         item_base = data_nodes[crep.data]
7397     end
7398
7399     if crep then
7400         step = crep.step or step
7401     end
7402
7403     if crep and crep.after then
7404         crep.insert = true
7405         if dummy_node then
7406             item = dummy_node
7407         else -- TODO. if there is a node after?
7408             d = node.copy(item_base)
7409             head, item = node.insert_after(head, item, d)
7410             dummy_node = item
7411         end
7412     end
7413
7414     if crep and not crep.after and dummy_node then
7415         node.remove(head, dummy_node)
7416         dummy_node = nil
7417

```

```

7418     end
7419
7420     if not enabled then
7421         last_match = save_last
7422         goto next
7423
7424     elseif crep and next(crep) == nil then -- = {}
7425         if step == 0 then
7426             last_match = save_last    -- Optimization
7427         else
7428             last_match = utf8.offset(w, sc+step)
7429         end
7430         goto next
7431
7432     elseif crep == nil or crep.remove then
7433         node.remove(head, item)
7434         table.remove(w_nodes, sc)
7435         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7436         sc = sc - 1 -- Nothing has been inserted.
7437         last_match = utf8.offset(w, sc+1+step)
7438         goto next
7439
7440     elseif crep and crep.kashida then -- Experimental
7441         node.set_attribute(item,
7442             Babel.attr_kashida,
7443             crep.kashida)
7444         last_match = utf8.offset(w, sc+1+step)
7445         goto next
7446
7447     elseif crep and crep.string then
7448         local str = crep.string(matches)
7449         if str == '' then -- Gather with nil
7450             node.remove(head, item)
7451             table.remove(w_nodes, sc)
7452             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7453             sc = sc - 1 -- Nothing has been inserted.
7454         else
7455             local loop_first = true
7456             for s in string.utfvalues(str) do
7457                 d = node.copy(item_base)
7458                 d.char = s
7459                 if loop_first then
7460                     loop_first = false
7461                     head, new = node.insert_before(head, item, d)
7462                     if sc == 1 then
7463                         word_head = head
7464                     end
7465                     w_nodes[sc] = d
7466                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7467                 else
7468                     sc = sc + 1
7469                     head, new = node.insert_before(head, item, d)
7470                     table.insert(w_nodes, sc, new)
7471                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7472                 end
7473                 if Babel.debug then
7474                     print('.....', 'str')
7475                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7476                 end
7477             end -- for
7478             node.remove(head, item)
7479         end -- if ''
7480         last_match = utf8.offset(w, sc+1+step)

```



```

7481         goto next
7482
7483     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7484         d = node.new(7, 3) -- (disc, regular)
7485         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7486         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7487         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7488         d.attr = item_base.attr
7489         if crep.pre == nil then -- TeXbook p96
7490             d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7491         else
7492             d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7493         end
7494         placeholder = '|'
7495         head, new = node.insert_before(head, item, d)
7496
7497     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7498         -- ERROR
7499
7500     elseif crep and crep.penalty then
7501         d = node.new(14, 0) -- (penalty, userpenalty)
7502         d.attr = item_base.attr
7503         d.penalty = tovalue(crep.penalty)
7504         head, new = node.insert_before(head, item, d)
7505
7506     elseif crep and crep.space then
7507         -- 655360 = 10 pt = 10 * 65536 sp
7508         d = node.new(12, 13) -- (glue, spaceskip)
7509         local quad = font.getfont(item_base.font).size or 655360
7510         node.setglue(d, tovalue(crep.space[1]) * quad,
7511                        tovalue(crep.space[2]) * quad,
7512                        tovalue(crep.space[3]) * quad)
7513         if mode == 0 then
7514             placeholder = ' '
7515         end
7516         head, new = node.insert_before(head, item, d)
7517
7518     elseif crep and crep.norule then
7519         -- 655360 = 10 pt = 10 * 65536 sp
7520         d = node.new(2, 3) -- (rule, empty) = \no*rule
7521         local quad = font.getfont(item_base.font).size or 655360
7522         d.width = tovalue(crep.norule[1]) * quad
7523         d.height = tovalue(crep.norule[2]) * quad
7524         d.depth = tovalue(crep.norule[3]) * quad
7525         head, new = node.insert_before(head, item, d)
7526
7527     elseif crep and crep.spacefactor then
7528         d = node.new(12, 13) -- (glue, spaceskip)
7529         local base_font = font.getfont(item_base.font)
7530         node.setglue(d,
7531                      tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7532                      tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7533                      tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7534         if mode == 0 then
7535             placeholder = ' '
7536         end
7537         head, new = node.insert_before(head, item, d)
7538
7539     elseif mode == 0 and crep and crep.space then
7540         -- ERROR
7541
7542     elseif crep and crep.kern then
7543         d = node.new(13, 1) -- (kern, user)

```

```

7544         local quad = font.getfont(item_base.font).size or 655360
7545         d.attr = item_base.attr
7546         d.kern = tovalue(crep.kern) * quad
7547         head, new = node.insert_before(head, item, d)
7548
7549     elseif crep and crep.node then
7550         d = node.new(crep.node[1], crep.node[2])
7551         d.attr = item_base.attr
7552         head, new = node.insert_before(head, item, d)
7553
7554     end -- i.e., replacement cases
7555
7556     -- Shared by disc, space(factor), kern, node and penalty.
7557     if sc == 1 then
7558         word_head = head
7559     end
7560     if crep.insert then
7561         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7562         table.insert(w_nodes, sc, new)
7563         last = last + 1
7564     else
7565         w_nodes[sc] = d
7566         node.remove(head, item)
7567         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7568     end
7569
7570     last_match = utf8.offset(w, sc+1+step)
7571
7572     ::next::
7573
7574     end -- for each replacement
7575
7576     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7577     if Babel.debug then
7578         print('.....', '/')
7579         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7580     end
7581
7582     if dummy_node then
7583         node.remove(head, dummy_node)
7584         dummy_node = nil
7585     end
7586
7587     end -- for match
7588
7589     end -- for patterns
7590
7591     ::next::
7592     word_head = nw
7593     end -- for substring
7594
7595     if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7596     return head
7597 end
7598
7599 -- This table stores capture maps, numbered consecutively
7600 Babel.capture_maps = {}
7601
7602 -- The following functions belong to the next macro
7603 function Babel.capture_func(key, cap)
7604     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
7605     local cnt
7606     local u = unicode.utf8

```

```

7607 ret, cnt = ret:gsub('{{([0-9])|([^\]|+)|(.-)}}', Babel.capture_func_map)
7608 if cnt == 0 then
7609     ret = u.gsub(ret, '{{(%X%X%X%X+}}',
7610         function (n)
7611             return u.char(tonumber(n, 16))
7612         end)
7613 end
7614 ret = ret:gsub("%[%[%]%.%.%", '')
7615 ret = ret:gsub("%.%.%.%[%[%]%", '')
7616 return key .. [[=function(m) return ]] .. ret .. [[ end]]
7617 end
7618
7619 function Babel.capt_map(from, mapno)
7620     return Babel.capture_maps[mapno][from] or from
7621 end
7622
7623 -- Handle the {n|abc|ABC} syntax in captures
7624 function Babel.capture_func_map(capno, from, to)
7625     local u = unicode.utf8
7626     from = u.gsub(from, '{{(%X%X%X%X+}}',
7627         function (n)
7628             return u.char(tonumber(n, 16))
7629         end)
7630     to = u.gsub(to, '{{(%X%X%X%X+}}',
7631         function (n)
7632             return u.char(tonumber(n, 16))
7633         end)
7634     local froms = {}
7635     for s in string.utfcharacters(from) do
7636         table.insert(froms, s)
7637     end
7638     local cnt = 1
7639     table.insert(Babel.capture_maps, {})
7640     local mlen = table.getn(Babel.capture_maps)
7641     for s in string.utfcharacters(to) do
7642         Babel.capture_maps[mlen][froms[cnt]] = s
7643         cnt = cnt + 1
7644     end
7645     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7646         (mlen) .. ").." .. "[["
7647 end
7648
7649 -- Create/Extend reversed sorted list of kashida weights:
7650 function Babel.capture_kashida(key, wt)
7651     wt = tonumber(wt)
7652     if Babel.kashida_wts then
7653         for p, q in ipairs(Babel.kashida_wts) do
7654             if wt == q then
7655                 break
7656             elseif wt > q then
7657                 table.insert(Babel.kashida_wts, p, wt)
7658                 break
7659             elseif table.getn(Babel.kashida_wts) == p then
7660                 table.insert(Babel.kashida_wts, wt)
7661             end
7662         end
7663     else
7664         Babel.kashida_wts = { wt }
7665     end
7666     return 'kashida = ' .. wt
7667 end
7668
7669 function Babel.capture_node(id, subtype)

```

```

7670 local sbt = 0
7671 for k, v in pairs(node.subtypes(id)) do
7672   if v == subtype then sbt = k end
7673 end
7674 return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7675 end
7676
7677 -- Experimental: applies prehyphenation transforms to a string (letters
7678 -- and spaces).
7679 function Babel.string_prehyphenation(str, locale)
7680   local n, head, last, res
7681   head = node.new(8, 0) -- dummy (hack just to start)
7682   last = head
7683   for s in string.utfvalues(str) do
7684     if s == 20 then
7685       n = node.new(12, 0)
7686     else
7687       n = node.new(29, 0)
7688       n.char = s
7689     end
7690     node.set_attribute(n, Babel.attr_locale, locale)
7691     last.next = n
7692     last = n
7693   end
7694   head = Babel.hyphenate_replace(head, 0)
7695   res = ''
7696   for n in node.traverse(head) do
7697     if n.id == 12 then
7698       res = res .. ' '
7699     elseif n.id == 29 then
7700       res = res .. unicode.utf8.char(n.char)
7701     end
7702   end
7703   tex.print(res)
7704 end
7705 </transforms>

```

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

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

```

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

```

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

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

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

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

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

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

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

```

7706 (*basic-r)
7707 Babel.bidi_enabled = true
7708
7709 require('babel-data-bidi.lua')
7710
7711 local characters = Babel.characters
7712 local ranges = Babel.ranges
7713
7714 local DIR = node.id("dir")
7715
7716 local function dir_mark(head, from, to, outer)
7717   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7718   local d = node.new(DIR)
7719   d.dir = '+' .. dir
7720   node.insert_before(head, from, d)
7721   d = node.new(DIR)
7722   d.dir = '-' .. dir
7723   node.insert_after(head, to, d)
7724 end
7725
7726 function Babel.bidi(head, ispar)
7727   local first_n, last_n          -- first and last char with nums
7728   local last_es                  -- an auxiliary 'last' used with nums
7729   local first_d, last_d          -- first and last char in L/R block
7730   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):

```

7731 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7732 local strong_lr = (strong == 'l') and 'l' or 'r'
7733 local outer = strong
7734
7735 local new_dir = false
7736 local first_dir = false
7737 local inmath = false
7738
7739 local last_lr
7740
7741 local type_n = ''
7742
7743 for item in node.traverse(head) do
7744
7745   -- three cases: glyph, dir, otherwise
7746   if item.id == node.id'glyph'
7747     or (item.id == 7 and item.subtype == 2) then
7748
7749     local itemchar
7750     if item.id == 7 and item.subtype == 2 then
7751       itemchar = item.replace.char

```

```

7752     else
7753         itemchar = item.char
7754     end
7755     local chardata = characters[itemchar]
7756     dir = chardata and chardata.d or nil
7757     if not dir then
7758         for nn, et in ipairs(ranges) do
7759             if itemchar < et[1] then
7760                 break
7761             elseif itemchar <= et[2] then
7762                 dir = et[3]
7763                 break
7764             end
7765         end
7766     end
7767     dir = dir or 'l'
7768     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).

```

7769     if new_dir then
7770         attr_dir = 0
7771         for at in node.traverse(item.attr) do
7772             if at.number == Babel.attr_dir then
7773                 attr_dir = at.value & 0x3
7774             end
7775         end
7776         if attr_dir == 1 then
7777             strong = 'r'
7778         elseif attr_dir == 2 then
7779             strong = 'al'
7780         else
7781             strong = 'l'
7782         end
7783         strong_lr = (strong == 'l') and 'l' or 'r'
7784         outer = strong_lr
7785         new_dir = false
7786     end
7787
7788     if dir == 'nsm' then dir = strong end -- W1

```

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

```

7789     dir_real = dir -- We need dir_real to set strong below
7790     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:

```

7791     if strong == 'al' then
7792         if dir == 'en' then dir = 'an' end -- W2
7793         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7794         strong_lr = 'r' -- W3
7795     end

```

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

```

7796     elseif item.id == node.id'dir' and not inmath then
7797         new_dir = true
7798         dir = nil
7799     elseif item.id == node.id'math' then
7800         inmath = (item.subtype == 0)
7801     else
7802         dir = nil -- Not a char
7803     end

```

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

```

7804   if dir == 'en' or dir == 'an' or dir == 'et' then
7805       if dir ~= 'et' then
7806           type_n = dir
7807       end
7808       first_n = first_n or item
7809       last_n = last_es or item
7810       last_es = nil
7811   elseif dir == 'es' and last_n then -- W3+W6
7812       last_es = item
7813   elseif dir == 'cs' then           -- it's right - do nothing
7814   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7815       if strong_lr == 'r' and type_n ~= '' then
7816           dir_mark(head, first_n, last_n, 'r')
7817       elseif strong_lr == 'l' and first_d and type_n == 'an' then
7818           dir_mark(head, first_n, last_n, 'r')
7819           dir_mark(head, first_d, last_d, outer)
7820           first_d, last_d = nil, nil
7821       elseif strong_lr == 'l' and type_n ~= '' then
7822           last_d = last_n
7823       end
7824       type_n = ''
7825       first_n, last_n = nil, nil
7826   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:

```

7827   if dir == 'l' or dir == 'r' then
7828       if dir ~= outer then
7829           first_d = first_d or item
7830           last_d = item
7831       elseif first_d and dir ~= strong_lr then
7832           dir_mark(head, first_d, last_d, outer)
7833           first_d, last_d = nil, nil
7834       end
7835   end

```

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

```

7836   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7837       item.char = characters[item.char] and
7838           characters[item.char].m or item.char
7839   elseif (dir or new_dir) and last_lr ~= item then
7840       local mir = outer .. strong_lr .. (dir or outer)
7841       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7842           for ch in node.traverse(node.next(last_lr)) do
7843               if ch == item then break end
7844               if ch.id == node.id'glyph' and characters[ch.char] then
7845                   ch.char = characters[ch.char].m or ch.char
7846               end
7847           end
7848       end
7849   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).

```

7850   if dir == 'l' or dir == 'r' then
7851       last_lr = item
7852       strong = dir_real          -- Don't search back - best save now
7853       strong_lr = (strong == 'l') and 'l' or 'r'
7854   elseif new_dir then
7855       last_lr = nil
7856   end
7857 end

```

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

```

7858   if last_lr and outer == 'r' then
7859       for ch in node.traverse_id(node.id('glyph', node.next(last_lr)) do
7860           if characters[ch.char] then
7861               ch.char = characters[ch.char].m or ch.char
7862           end
7863       end
7864   end
7865   if first_n then
7866       dir_mark(head, first_n, last_n, outer)
7867   end
7868   if first_d then
7869       dir_mark(head, first_d, last_d, outer)
7870   end

```

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

```

7871   return node.prev(head) or head
7872 end
7873 </basic-r>

```

And here the Lua code for bidi=basic:

```

7874 <{*basic>
7875 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7876
7877 Babel.fontmap = Babel.fontmap or {}
7878 Babel.fontmap[0] = {}          -- l
7879 Babel.fontmap[1] = {}          -- r
7880 Babel.fontmap[2] = {}          -- al/an
7881
7882 -- To cancel mirroring. Also OML, OMS, U?
7883 Babel.symbol_fonts = Babel.symbol_fonts or {}
7884 Babel.symbol_fonts[font.id('tenln')] = true
7885 Babel.symbol_fonts[font.id('tenlnw')] = true
7886 Babel.symbol_fonts[font.id('tencirc')] = true
7887 Babel.symbol_fonts[font.id('tencircw')] = true
7888
7889 Babel.bidi_enabled = true
7890 Babel.mirroring_enabled = true
7891
7892 require('babel-data-bidi.lua')
7893
7894 local characters = Babel.characters
7895 local ranges = Babel.ranges
7896
7897 local DIR = node.id('dir')
7898 local GLYPH = node.id('glyph')
7899
7900 local function insert_implicit(head, state, outer)
7901     local new_state = state
7902     if state.sim and state.eim and state.sim ~= state.eim then
7903         dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse

```



```

7904     local d = node.new(DIR)
7905     d.dir = '+' .. dir
7906     node.insert_before(head, state.sim, d)
7907     local d = node.new(DIR)
7908     d.dir = '-' .. dir
7909     node.insert_after(head, state.eim, d)
7910 end
7911 new_state.sim, new_state.eim = nil, nil
7912 return head, new_state
7913 end
7914
7915 local function insert_numeric(head, state)
7916     local new
7917     local new_state = state
7918     if state.san and state.ean and state.san ~= state.ean then
7919         local d = node.new(DIR)
7920         d.dir = '+TLT'
7921         _, new = node.insert_before(head, state.san, d)
7922         if state.san == state.sim then state.sim = new end
7923         local d = node.new(DIR)
7924         d.dir = '-TLT'
7925         _, new = node.insert_after(head, state.ean, d)
7926         if state.ean == state.eim then state.eim = new end
7927     end
7928     new_state.san, new_state.ean = nil, nil
7929     return head, new_state
7930 end
7931
7932 local function glyph_not_symbol_font(node)
7933     if node.id == GLYPH then
7934         return not Babel.symbol_fonts[node.font]
7935     else
7936         return false
7937     end
7938 end
7939
7940 -- TODO - \hbox with an explicit dir can lead to wrong results
7941 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7942 -- was made to improve the situation, but the problem is the 3-dir
7943 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7944 -- well.
7945
7946 function Babel.bidi(head, ispar, hdir)
7947     local d -- d is used mainly for computations in a loop
7948     local prev_d = ''
7949     local new_d = false
7950
7951     local nodes = {}
7952     local outer_first = nil
7953     local inmath = false
7954
7955     local glue_d = nil
7956     local glue_i = nil
7957
7958     local has_en = false
7959     local first_et = nil
7960
7961     local has_hyperlink = false
7962
7963     local ATDIR = Babel.attr_dir
7964     local attr_d, temp
7965     local locale_d
7966

```

```

7967 local save_outer
7968 local locale_d = node.get_attribute(head, ATDIR)
7969 if locale_d then
7970     locale_d = locale_d & 0x3
7971     save_outer = (locale_d == 0 and 'l') or
7972                  (locale_d == 1 and 'r') or
7973                  (locale_d == 2 and 'al')
7974 elseif ispar then      -- Or error? Shouldn't happen
7975     -- when the callback is called, we are just _after_ the box,
7976     -- and the textdir is that of the surrounding text
7977     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7978 else                  -- Empty box
7979     save_outer = ('TRT' == hdir) and 'r' or 'l'
7980 end
7981 local outer = save_outer
7982 local last = outer
7983 -- 'al' is only taken into account in the first, current loop
7984 if save_outer == 'al' then save_outer = 'r' end
7985
7986 local fontmap = Babel.fontmap
7987
7988 for item in node.traverse(head) do
7989
7990     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
7991     locale_d = node.get_attribute(item, ATDIR)
7992     node.set_attribute(item, ATDIR, 0x80)
7993
7994     -- In what follows, #node is the last (previous) node, because the
7995     -- current one is not added until we start processing the neutrals.
7996     -- three cases: glyph, dir, otherwise
7997     if glyph_not_symbol_font(item)
7998         or (item.id == 7 and item.subtype == 2) then
7999
8000         if locale_d == 0x80 then goto nextnode end
8001
8002         local d_font = nil
8003         local item_r
8004         if item.id == 7 and item.subtype == 2 then
8005             item_r = item.replace      -- automatic discs have just 1 glyph
8006         else
8007             item_r = item
8008         end
8009
8010         local chardata = characters[item_r.char]
8011         d = chardata and chardata.d or nil
8012         if not d or d == 'nsm' then
8013             for nn, et in ipairs(ranges) do
8014                 if item_r.char < et[1] then
8015                     break
8016                 elseif item_r.char <= et[2] then
8017                     if not d then d = et[3]
8018                     elseif d == 'nsm' then d_font = et[3]
8019                     end
8020                     break
8021                 end
8022             end
8023         end
8024         d = d or 'l'
8025
8026         -- A short 'pause' in bidi for mapfont
8027         -- %%% TODO. move if fontmap here
8028         d_font = d_font or d
8029         d_font = (d_font == 'l' and 0) or

```

```

8030             (d_font == 'nsm' and 0) or
8031             (d_font == 'r' and 1) or
8032             (d_font == 'al' and 2) or
8033             (d_font == 'an' and 2) or nil
8034 if d_font and fontmap and fontmap[d_font][item_r.font] then
8035     item_r.font = fontmap[d_font][item_r.font]
8036 end
8037
8038 if new_d then
8039     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8040     if inmath then
8041         attr_d = 0
8042     else
8043         attr_d = locale_d & 0x3
8044     end
8045     if attr_d == 1 then
8046         outer_first = 'r'
8047         last = 'r'
8048     elseif attr_d == 2 then
8049         outer_first = 'r'
8050         last = 'al'
8051     else
8052         outer_first = 'l'
8053         last = 'l'
8054     end
8055     outer = last
8056     has_en = false
8057     first_et = nil
8058     new_d = false
8059 end
8060
8061 if glue_d then
8062     if (d == 'l' and 'l' or 'r') ~= glue_d then
8063         table.insert(nodes, {glue_i, 'on', nil})
8064     end
8065     glue_d = nil
8066     glue_i = nil
8067 end
8068
8069 elseif item.id == DIR then
8070     d = nil
8071     new_d = true
8072
8073 elseif item.id == node.id'glue' and item.subtype == 13 then
8074     glue_d = d
8075     glue_i = item
8076     d = nil
8077
8078 elseif item.id == node.id'math' then
8079     inmath = (item.subtype == 0)
8080
8081 elseif item.id == 8 and item.subtype == 19 then
8082     has_hyperlink = true
8083
8084 else
8085     d = nil
8086 end
8087
8088 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8089 if last == 'al' and d == 'en' then
8090     d = 'an'          -- W3
8091 elseif last == 'al' and (d == 'et' or d == 'es') then
8092     d = 'on'          -- W6

```

```

8093 end
8094
8095 -- EN + CS/ES + EN      -- W4
8096 if d == 'en' and #nodes >= 2 then
8097   if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8098     and nodes[#nodes-1][2] == 'en' then
8099     nodes[#nodes][2] = 'en'
8100   end
8101 end
8102
8103 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8104 if d == 'an' and #nodes >= 2 then
8105   if (nodes[#nodes][2] == 'cs')
8106     and nodes[#nodes-1][2] == 'an' then
8107     nodes[#nodes][2] = 'an'
8108   end
8109 end
8110
8111 -- ET/EN                  -- W5 + W7->l / W6->on
8112 if d == 'et' then
8113   first_et = first_et or (#nodes + 1)
8114 elseif d == 'en' then
8115   has_en = true
8116   first_et = first_et or (#nodes + 1)
8117 elseif first_et then      -- d may be nil here !
8118   if has_en then
8119     if last == 'l' then
8120       temp = 'l'      -- W7
8121     else
8122       temp = 'en'     -- W5
8123     end
8124   else
8125     temp = 'on'       -- W6
8126   end
8127   for e = first_et, #nodes do
8128     if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8129   end
8130   first_et = nil
8131   has_en = false
8132 end
8133
8134 -- Force mathdir in math if ON (currently works as expected only
8135 -- with 'l')
8136
8137 if inmath and d == 'on' then
8138   d = ('TRT' == tex.mathdir) and 'r' or 'l'
8139 end
8140
8141 if d then
8142   if d == 'al' then
8143     d = 'r'
8144     last = 'al'
8145   elseif d == 'l' or d == 'r' then
8146     last = d
8147   end
8148   prev_d = d
8149   table.insert(nodes, {item, d, outer_first})
8150 end
8151
8152 outer_first = nil
8153
8154 ::nextnode::
8155

```

```

8156 end -- for each node
8157
8158 -- TODO -- repeated here in case EN/ET is the last node. Find a
8159 -- better way of doing things:
8160 if first_et then -- dir may be nil here !
8161   if has_en then
8162     if last == 'l' then
8163       temp = 'l' -- W7
8164     else
8165       temp = 'en' -- W5
8166     end
8167   else
8168     temp = 'on' -- W6
8169   end
8170   for e = first_et, #nodes do
8171     if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8172   end
8173 end
8174
8175 -- dummy node, to close things
8176 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8177
8178 ----- NEUTRAL -----
8179
8180 outer = save_outer
8181 last = outer
8182
8183 local first_on = nil
8184
8185 for q = 1, #nodes do
8186   local item
8187
8188   local outer_first = nodes[q][3]
8189   outer = outer_first or outer
8190   last = outer_first or last
8191
8192   local d = nodes[q][2]
8193   if d == 'an' or d == 'en' then d = 'r' end
8194   if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8195
8196   if d == 'on' then
8197     first_on = first_on or q
8198   elseif first_on then
8199     if last == d then
8200       temp = d
8201     else
8202       temp = outer
8203     end
8204     for r = first_on, q - 1 do
8205       nodes[r][2] = temp
8206       item = nodes[r][1] -- MIRRORING
8207       if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8208         and temp == 'r' and characters[item.char] then
8209         local font_mode = ''
8210         if item.font > 0 and font.fonts[item.font].properties then
8211           font_mode = font.fonts[item.font].properties.mode
8212         end
8213         if font_mode ~= 'harf' and font_mode ~= 'plug' then
8214           item.char = characters[item.char].m or item.char
8215         end
8216       end
8217     end
8218     first_on = nil

```

```

8219     end
8220
8221     if d == 'r' or d == 'l' then last = d end
8222 end
8223
8224 ----- IMPLICIT, REORDER -----
8225
8226 outer = save_outer
8227 last = outer
8228
8229 local state = {}
8230 state.has_r = false
8231
8232 for q = 1, #nodes do
8233
8234     local item = nodes[q][1]
8235
8236     outer = nodes[q][3] or outer
8237
8238     local d = nodes[q][2]
8239
8240     if d == 'nsm' then d = last end          -- W1
8241     if d == 'en' then d = 'an' end
8242     local isdir = (d == 'r' or d == 'l')
8243
8244     if outer == 'l' and d == 'an' then
8245         state.san = state.san or item
8246         state.ean = item
8247     elseif state.san then
8248         head, state = insert_numeric(head, state)
8249     end
8250
8251     if outer == 'l' then
8252         if d == 'an' or d == 'r' then      -- im -> implicit
8253             if d == 'r' then state.has_r = true end
8254             state.sim = state.sim or item
8255             state.eim = item
8256         elseif d == 'l' and state.sim and state.has_r then
8257             head, state = insert_implicit(head, state, outer)
8258         elseif d == 'l' then
8259             state.sim, state.eim, state.has_r = nil, nil, false
8260         end
8261     else
8262         if d == 'an' or d == 'l' then
8263             if nodes[q][3] then -- nil except after an explicit dir
8264                 state.sim = item -- so we move sim 'inside' the group
8265             else
8266                 state.sim = state.sim or item
8267             end
8268             state.eim = item
8269         elseif d == 'r' and state.sim then
8270             head, state = insert_implicit(head, state, outer)
8271         elseif d == 'r' then
8272             state.sim, state.eim = nil, nil
8273         end
8274     end
8275
8276     if isdir then
8277         last = d          -- Don't search back - best save now
8278     elseif d == 'on' and state.san then
8279         state.san = state.san or item
8280         state.ean = item
8281     end

```

```

8282
8283 end
8284
8285 head = node.prev(head) or head
8286 % \end{macrocode}
8287 %
8288 % Now direction nodes has been distributed with relation to characters
8289 % and spaces, we need to take into account \TeX-specific elements in
8290 % the node list, to move them at an appropriate place. Firstly, with
8291 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8292 % that the latter are still discardable.
8293 %
8294 % \begin{macrocode}
8295 --- FIXES ---
8296 if has_hyperlink then
8297   local flag, linking = 0, 0
8298   for item in node.traverse(head) do
8299     if item.id == DIR then
8300       if item.dir == '+TRT' or item.dir == '+TLT' then
8301         flag = flag + 1
8302       elseif item.dir == '-TRT' or item.dir == '-TLT' then
8303         flag = flag - 1
8304       end
8305       elseif item.id == 8 and item.subtype == 19 then
8306         linking = flag
8307       elseif item.id == 8 and item.subtype == 20 then
8308         if linking > 0 then
8309           if item.prev.id == DIR and
8310             (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8311             d = node.new(DIR)
8312             d.dir = item.prev.dir
8313             node.remove(head, item.prev)
8314             node.insert_after(head, item, d)
8315           end
8316         end
8317         linking = 0
8318       end
8319     end
8320   end
8321
8322   for item in node.traverse_id(10, head) do
8323     local p = item
8324     local flag = false
8325     while p.prev and p.prev.id == 14 do
8326       flag = true
8327       p = p.prev
8328     end
8329     if flag then
8330       node.insert_before(head, p, node.copy(item))
8331       node.remove(head, item)
8332     end
8333   end
8334
8335   return head
8336 end
8337
8337 function Babel.unset_atdir(head)
8338   local ATDIR = Babel.attr_dir
8339   for item in node.traverse(head) do
8340     node.set_attribute(item, ATDIR, 0x80)
8341   end
8342   return head
8343 end
8344 </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.

```
8345 (*nil)
8346 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8347 \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.

```
8348 \ifx\l@nil\undefined
8349 \newlanguage\l@nil
8350 \@namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8351 \let\bbl@elt\relax
8352 \edef\bbl@languages{% Add it to the list of languages
8353 \bbl@languages\bbl@elt{nil}{the\l@nil}{}}
8354 \fi
```

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

```
8355 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

**`\captionnil`**  
**`\datenil`**

```
8356 \let\captionnil\@empty
8357 \let\datenil\@empty
```

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

```
8358 \def\bbl@inidata@nil{%
8359 \bbl@elt{identification}{tag.ini}{und}%
8360 \bbl@elt{identification}{load.level}{0}%
8361 \bbl@elt{identification}{charset}{utf8}%
8362 \bbl@elt{identification}{version}{1.0}%
8363 \bbl@elt{identification}{date}{2022-05-16}%
8364 \bbl@elt{identification}{name.local}{nil}%
8365 \bbl@elt{identification}{name.english}{nil}%
8366 \bbl@elt{identification}{name.babel}{nil}%
8367 \bbl@elt{identification}{tag.bcp47}{und}%
8368 \bbl@elt{identification}{language.tag.bcp47}{und}%
8369 \bbl@elt{identification}{tag.opentype}{dfLT}%
8370 \bbl@elt{identification}{script.name}{Latin}%
8371 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8372 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8373 \bbl@elt{identification}{level}{1}%
```



```

8374 \bbl@elt{identification}{encodings}{}%
8375 \bbl@elt{identification}{derivate}{no}}
8376 \@namedef{bbl@tbc@nil}{und}
8377 \@namedef{bbl@lbc@nil}{und}
8378 \@namedef{bbl@casing@nil}{und}
8379 \@namedef{bbl@lotf@nil}{dflt}
8380 \@namedef{bbl@elname@nil}{nil}
8381 \@namedef{bbl@lname@nil}{nil}
8382 \@namedef{bbl@esname@nil}{Latin}
8383 \@namedef{bbl@sname@nil}{Latin}
8384 \@namedef{bbl@sbc@nil}{Latn}
8385 \@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.

```

8386 \ldf@finish{nil}
8387 </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.

```

8388 <<Compute Julian day>> ≡
8389 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8390 \def\bbl@cs@gregleap#1{%
8391   (\bbl@fpmo{#1}{4} == 0) &&
8392   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
8393 \def\bbl@cs@jd#1#2#3{% year, month, day
8394   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8395     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8396     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8397     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8398 <</Compute Julian day>>

```

### 13.1. Islamic

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

```

8399 <*ca-islamic>
8400 \ExplSyntaxOn
8401 <@Compute Julian day@>
8402 % == islamic (default)
8403 % Not yet implemented
8404 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8405 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8406   ((#3 + ceil(29.5 * (#2 - 1)) +
8407     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8408     1948439.5) - 1) }
8409 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8410 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8411 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8412 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8413 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8414 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8415   \edef\bbl@tempa{%
8416     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 }%
8417   }
8418   \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) } }%
8419   \edef#6{\fp_eval:n{

```

```

8420     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8421 \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).

```

8422 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8423 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8424 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8425 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8426 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8427 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8428 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8429 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8430 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8431 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8432 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8433 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8434 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8435 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8436 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8437 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8438 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8439 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8440 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8441 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8442 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8443 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8444 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8445 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8446 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8447 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8448 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8449 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8450 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8451 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8452 65401,65431,65460,65490,65520}
8453 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8454 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8455 \namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8456 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8457 \ifnum#2>2014 \ifnum#2<2038
8458 \bbl@afterfi\expandafter\@gobble
8459 \fi\fi
8460 {\bbl@error{year-out-range}{2014-2038}}{}}%
8461 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8462 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8463 \count@\@ne
8464 \bbl@foreach\bbl@cs@umalqura@data{%
8465 \advance\count@\@ne
8466 \ifnum##1>\bbl@tempd\else
8467 \edef\bbl@tempe{\the\count@}%
8468 \edef\bbl@tempb{##1}%
8469 \fi}%
8470 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
8471 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8472 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8473 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8474 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8475 \ExplSyntaxOff
8476 \bbl@add\bbl@precalendar{%
8477 \bbl@replace\bbl@ld@calendar{-civil}}}%

```

```

8478 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
8479 \bbl@replace\bbl@ld@calendar{+}{}%
8480 \bbl@replace\bbl@ld@calendar{-}{}%
8481 </ca-islamic>

```

## 13.2. Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaption 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`

```

8482 <*/ca-hebrew>
8483 \newcount\bbl@cntcommon
8484 \def\bbl@remainder#1#2#3{%
8485   #3=#1\relax
8486   \divide #3 by #2\relax
8487   \multiply #3 by -#2\relax
8488   \advance #3 by #1\relax}%
8489 \newif\ifbbl@divisible
8490 \def\bbl@checkifdivisible#1#2{%
8491   {\countdef\tmp=0
8492     \bbl@remainder{#1}{#2}{\tmp}%
8493     \ifnum \tmp=0
8494       \global\bbl@divisibletrue
8495     \else
8496       \global\bbl@divisiblefalse
8497     \fi}}
8498 \newif\ifbbl@gregleap
8499 \def\bbl@ifgregleap#1{%
8500   \bbl@checkifdivisible{#1}{4}%
8501   \ifbbl@divisible
8502     \bbl@checkifdivisible{#1}{100}%
8503     \ifbbl@divisible
8504       \bbl@checkifdivisible{#1}{400}%
8505       \ifbbl@divisible
8506         \bbl@gregleaptrue
8507       \else
8508         \bbl@gregleapfalse
8509       \fi
8510     \else
8511       \bbl@gregleaptrue
8512     \fi
8513   \else
8514     \bbl@gregleapfalse
8515   \fi
8516   \ifbbl@gregleap}
8517 \def\bbl@gregdayspriormonths#1#2#3{%
8518   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8519     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8520   \bbl@ifgregleap{#2}%
8521   \ifnum #1 > 2
8522     \advance #3 by 1
8523   \fi
8524   \fi
8525   \global\bbl@cntcommon=#3}%
8526   #3=\bbl@cntcommon}
8527 \def\bbl@gregdaysprioryears#1#2{%
8528   {\countdef\tmpc=4
8529     \countdef\tmpb=2
8530     \tmpb=#1\relax
8531     \advance \tmpb by -1
8532     \tmpc=\tmpb
8533     \multiply \tmpc by 365
8534     #2=\tmpc

```

```

8535 \tmpc=\tmpb
8536 \divide \tmpc by 4
8537 \advance #2 by \tmpc
8538 \tmpc=\tmpb
8539 \divide \tmpc by 100
8540 \advance #2 by -\tmpc
8541 \tmpc=\tmpb
8542 \divide \tmpc by 400
8543 \advance #2 by \tmpc
8544 \global\bbl@cntcommon=#2\relax}%
8545 #2=\bbl@cntcommon}
8546 \def\bbl@absfromgreg#1#2#3#4{%
8547 {\countdef\tmpd=0
8548 #4=#1\relax
8549 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8550 \advance #4 by \tmpd
8551 \bbl@gregdaysprioryears{#3}{\tmpd}%
8552 \advance #4 by \tmpd
8553 \global\bbl@cntcommon=#4\relax}%
8554 #4=\bbl@cntcommon}
8555 \newif\ifbbl@hebrleap
8556 \def\bbl@checkleaphebrewyear#1{%
8557 {\countdef\tmpa=0
8558 \countdef\tmpb=1
8559 \tmpa=#1\relax
8560 \multiply \tmpa by 7
8561 \advance \tmpa by 1
8562 \bbl@remainder{\tmpa}{19}{\tmpb}%
8563 \ifnum \tmpb < 7
8564 \global\bbl@hebrleaptrue
8565 \else
8566 \global\bbl@hebrleapfalse
8567 \fi}}
8568 \def\bbl@hebreleapsedmonths#1#2{%
8569 {\countdef\tmpa=0
8570 \countdef\tmpb=1
8571 \countdef\tmpc=2
8572 \tmpa=#1\relax
8573 \advance \tmpa by -1
8574 #2=\tmpa
8575 \divide #2 by 19
8576 \multiply #2 by 235
8577 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8578 \tmpc=\tmpb
8579 \multiply \tmpb by 12
8580 \advance #2 by \tmpb
8581 \multiply \tmpc by 7
8582 \advance \tmpc by 1
8583 \divide \tmpc by 19
8584 \advance #2 by \tmpc
8585 \global\bbl@cntcommon=#2}%
8586 #2=\bbl@cntcommon}
8587 \def\bbl@hebreleapseddays#1#2{%
8588 {\countdef\tmpa=0
8589 \countdef\tmpb=1
8590 \countdef\tmpc=2
8591 \bbl@hebreleapsedmonths{#1}{#2}%
8592 \tmpa=#2\relax
8593 \multiply \tmpa by 13753
8594 \advance \tmpa by 5604
8595 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8596 \divide \tmpa by 25920
8597 \multiply #2 by 29

```

```

8598 \advance #2 by 1
8599 \advance #2 by \tmpa
8600 \bbl@remainder{#2}{7}{\tmpa}%
8601 \ifnum \tmpc < 19440
8602     \ifnum \tmpc < 9924
8603     \else
8604         \ifnum \tmpa=2
8605             \bbl@checkleaphebrewyear{#1}% of a common year
8606             \ifbbl@hebrleap
8607             \else
8608                 \advance #2 by 1
8609             \fi
8610         \fi
8611     \fi
8612     \ifnum \tmpc < 16789
8613     \else
8614         \ifnum \tmpa=1
8615             \advance #1 by -1
8616             \bbl@checkleaphebrewyear{#1}% at the end of leap year
8617             \ifbbl@hebrleap
8618                 \advance #2 by 1
8619             \fi
8620         \fi
8621     \fi
8622 \else
8623     \advance #2 by 1
8624 \fi
8625 \bbl@remainder{#2}{7}{\tmpa}%
8626 \ifnum \tmpa=0
8627     \advance #2 by 1
8628 \else
8629     \ifnum \tmpa=3
8630         \advance #2 by 1
8631     \else
8632         \ifnum \tmpa=5
8633             \advance #2 by 1
8634         \fi
8635     \fi
8636 \fi
8637 \global\bbl@cntcommon=#2\relax}%
8638 #2=\bbl@cntcommon}
8639 \def\bbl@daysinhebrewyear#1#2{%
8640 {\countdef\tmpe=12
8641 \bbl@hebreleapseddays{#1}{\tmpe}%
8642 \advance #1 by 1
8643 \bbl@hebreleapseddays{#1}{#2}%
8644 \advance #2 by -\tmpe
8645 \global\bbl@cntcommon=#2}%
8646 #2=\bbl@cntcommon}
8647 \def\bbl@hebrdayspriormonths#1#2#3{%
8648 {\countdef\tmpf= 14
8649 #3=\ifcase #1
8650     0 \or
8651     0 \or
8652     30 \or
8653     59 \or
8654     89 \or
8655     118 \or
8656     148 \or
8657     148 \or
8658     177 \or
8659     207 \or
8660     236 \or

```

```

8661         266 \or
8662         295 \or
8663         325 \or
8664         400
8665     \fi
8666     \bbl@checkleaphebyear{#2}%
8667     \ifbbl@hebrleap
8668         \ifnum #1 > 6
8669             \advance #3 by 30
8670         \fi
8671     \fi
8672     \bbl@daysinhebyear{#2}{\tmpf}%
8673     \ifnum #1 > 3
8674         \ifnum \tmpf=353
8675             \advance #3 by -1
8676         \fi
8677         \ifnum \tmpf=383
8678             \advance #3 by -1
8679         \fi
8680     \fi
8681     \ifnum #1 > 2
8682         \ifnum \tmpf=355
8683             \advance #3 by 1
8684         \fi
8685         \ifnum \tmpf=385
8686             \advance #3 by 1
8687         \fi
8688     \fi
8689     \global\bbl@cntcommon=#3\relax}%
8690     #3=\bbl@cntcommon}
8691 \def\bbl@absfromhebr#1#2#3#4{%
8692     {#4=#1\relax
8693     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8694     \advance #4 by #1\relax
8695     \bbl@hebrrelapseddays{#3}{#1}%
8696     \advance #4 by #1\relax
8697     \advance #4 by -1373429
8698     \global\bbl@cntcommon=#4\relax}%
8699     #4=\bbl@cntcommon}
8700 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8701     {\countdef\tmpx= 17
8702     \countdef\tmpy= 18
8703     \countdef\tmpz= 19
8704     #6=#3\relax
8705     \global\advance #6 by 3761
8706     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8707     \tmpz=1 \tmpy=1
8708     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8709     \ifnum \tmpx > #4\relax
8710         \global\advance #6 by -1
8711         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8712     \fi
8713     \advance #4 by -\tmpx
8714     \advance #4 by 1
8715     #5=#4\relax
8716     \divide #5 by 30
8717     \loop
8718         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8719         \ifnum \tmpx < #4\relax
8720             \advance #5 by 1
8721             \tmpy=\tmpx
8722         \repeat
8723     \global\advance #5 by -1

```

```

8724 \global\advance #4 by -\tmpy}}
8725 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8726 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8727 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8728 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8729 \bbl@hebrfromgreg
8730 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8731 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8732 \edef#4{\the\bbl@hebryear}%
8733 \edef#5{\the\bbl@hebrmonth}%
8734 \edef#6{\the\bbl@hebrday}}
8735 </ca-hebrew>

```

### 13.3. Persian

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

```

8736 <*ca-persian>
8737 \ExplSyntaxOn
8738 <@Compute Julian day@>
8739 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8740 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8741 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8742 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8743 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8744 \bbl@afterfi\expandafter\@gobble
8745 \fi\fi
8746 {\bbl@error{year-out-range}{2013-2050}{}}}%
8747 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8748 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8749 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8750 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8751 \ifnum\bbl@tempc<\bbl@tempb
8752 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8753 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8754 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8755 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8756 \fi
8757 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8758 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8759 \edef#5{\fp_eval:n{% set Jalali month
8760 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8761 \edef#6{\fp_eval:n{% set Jalali day
8762 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
8763 \ExplSyntaxOff
8764 </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.

```

8765 <*ca-coptic>
8766 \ExplSyntaxOn
8767 <@Compute Julian day@>
8768 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8769 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8770 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8771 \edef#4{\fp_eval:n{%
8772 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%

```

```

8773 \edef\bbl@tempc{\fp_eval:n{%
8774 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8775 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8776 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8777 \ExplSyntaxOff
8778 </ca-coptic>
8779 <*ca-ethiopic>
8780 \ExplSyntaxOn
8781 <@Compute Julian day@>
8782 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8783 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8784 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8785 \edef#4{\fp_eval:n{%
8786 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8787 \edef\bbl@tempc{\fp_eval:n{%
8788 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8789 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8790 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8791 \ExplSyntaxOff
8792 </ca-ethiopic>

```

### 13.5. Buddhist

That's very simple.

```

8793 <*ca-buddhist>
8794 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8795 \edef#4{\number\numexpr#1+543\relax}%
8796 \edef#5{#2}%
8797 \edef#6{#3}}
8798 </ca-buddhist>
8799 %
8800 % \subsection{Chinese}
8801 %
8802 % Brute force, with the Julian day of first day of each month. The
8803 % table has been computed with the help of \textsf{python-lunardate} by
8804 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8805 % is 2015-2044.
8806 %
8807 % \begin{macrocode}
8808 <*ca-chinese>
8809 \ExplSyntaxOn
8810 <@Compute Julian day@>
8811 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8812 \edef\bbl@tempd{\fp_eval:n{%
8813 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8814 \count@ \z@
8815 \@tempcnta=2015
8816 \bbl@foreach\bbl@cs@chinese@data{%
8817 \ifnum##1>\bbl@tempd\else
8818 \advance\count@\@ne
8819 \ifnum\count@>12
8820 \count@\@ne
8821 \advance\@tempcnta\@ne\fi
8822 \bbl@xin@{,##1,},{,\bbl@cs@chinese@leap,}%
8823 \ifin@
8824 \advance\count@\m@ne
8825 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8826 \else
8827 \edef\bbl@tempe{\the\count@}%
8828 \fi
8829 \edef\bbl@tempb{##1}%
8830 \fi}%
8831 \edef#4{\the\@tempcnta}%

```



```

8832 \edef#5{\bbl@tempe}%
8833 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8834 \def\bbl@cs@chinese@leap{%
8835 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8836 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8837 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8838 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8839 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8840 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8841 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8842 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8843 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8844 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8845 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8846 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8847 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8848 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8849 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8850 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8851 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8852 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8853 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8854 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8855 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8856 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8857 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8858 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8859 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8860 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8861 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8862 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8863 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8864 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8865 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8866 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8867 10896,10926,10956,10986,11015,11045,11074,11103}
8868 \ExplSyntaxOff
8869 </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`.

```

8870 <{*bplain | blplain}
8871 \catcode`\{=1 % left brace is begin-group character
8872 \catcode`\}=2 % right brace is end-group character
8873 \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).

```
8874 \openin 0 hyphen.cfg
8875 \ifeof0
8876 \else
8877 \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.

```
8878 \def\input #1 {%
8879 \let\input\input
8880 \a hyphen.cfg
8881 \let\input\input
8882 }
8883 \fi
8884 \bblatex\blatex
```

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

```
8885 \bblatex\input plain.tex
8886 \bblatex\input 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.

```
8887 \bblatex\def\fmtname{babel-plain}
8888 \bblatex\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).

```
8889 \bblatex\def\EmulateLaTeX\equiv
8890 \def\@empty{}
8891 \def\loadlocalcfg#1{%
8892 \openin0#1.cfg
8893 \ifeof0
8894 \closein0
8895 \else
8896 \closein0
8897 {\immediate\write16{*****}%
8898 \immediate\write16{* Local config file #1.cfg used}%
8899 \immediate\write16{**}%
8900 }
8901 \input #1.cfg\relax
8902 \fi
8903 \endofldef}
```

## 14.3. General tools

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

```
8904 \long\def\@firstofone#1{#1}
8905 \long\def\@firstoftwo#1#2{#1}
8906 \long\def\@secondoftwo#1#2{#2}
8907 \def\@nnil{\nil}
8908 \def\@gobbletwo#1#2{}
8909 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
```

```

8910 \def\@star@or@long#1{%
8911   \ifstar
8912     {\let\l@ngrel@x\relax#1}%
8913     {\let\l@ngrel@x\long#1}}
8914 \let\l@ngrel@x\relax
8915 \def\@car#1#2\@nil{#1}
8916 \def\@cdr#1#2\@nil{#2}
8917 \let\@typeset@protect\relax
8918 \let\protected@edef\edef
8919 \long\def\@gobble#1{}
8920 \edef\@backslashchar{\expandafter\@gobble\string\}
8921 \def\strip@prefix#1>{}
8922 \def\g@addto@macro#1#2{%
8923   \toks@\expandafter{#1#2}%
8924   \xdef#1{\the\toks@}}
8925 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8926 \def\@nameuse#1{\csname #1\endcsname}
8927 \def\@ifundefined#1{%
8928   \expandafter\ifx\csname#1\endcsname\relax
8929     \expandafter\@firstoftwo
8930   \else
8931     \expandafter\@secondoftwo
8932   \fi}
8933 \def\@expandtwoargs#1#2#3{%
8934   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8935 \def\zap@space#1 #2{%
8936   #1%
8937   \ifx#2\@empty\else\expandafter\zap@space\fi
8938   #2}
8939 \let\bbl@trace\@gobble
8940 \def\bbl@error#1{% Implicit #2#3#4
8941   \begingroup
8942     \catcode`\=0   \catcode`\==12 \catcode`\`=12
8943     \catcode`\^M=5 \catcode`\%=14
8944     \input errbabel.def
8945   \endgroup
8946   \bbl@error{#1}}
8947 \def\bbl@warning#1{%
8948   \begingroup
8949     \newlinechar=`^^J
8950     \def\{^^J(babel) }%
8951     \message{\{#1}%
8952   \endgroup}
8953 \let\bbl@infowarn\bbl@warning
8954 \def\bbl@info#1{%
8955   \begingroup
8956     \newlinechar=`^^J
8957     \def\{^^J}%
8958     \wlog{#1}%
8959   \endgroup}

```

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

```

8960 \ifx\@preamblecmds\undefined
8961   \def\@preamblecmds{}
8962 \fi
8963 \def\@onlypreamble#1{%
8964   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8965     \@preamblecmds\do#1}}
8966 \@onlypreamble\@onlypreamble

```

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

```

8967 \def\begindocument{%
8968   \@begindocumenthook

```

```

8969 \global\let\@begindocumenthook\@undefined
8970 \def\do##1{\global\let##1\@undefined}%
8971 \@preamblecmds
8972 \global\let\do\noexpand}

8973 \ifx\@begindocumenthook\@undefined
8974 \def\@begindocumenthook{}
8975 \fi
8976 \@onlypreamble\@begindocumenthook
8977 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8978 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8979 \@onlypreamble\AtEndOfPackage
8980 \def\@endoflfd{}
8981 \@onlypreamble\@endoflfd
8982 \let\bbl@afterlang\@empty
8983 \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.

```

8984 \catcode`\&=\z@
8985 \ifx&\if@files\@undefined
8986 \expandafter\let\csname if@files\expandafter\endcsname
8987 \csname iffalse\endcsname
8988 \fi
8989 \catcode`\&=4

```

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

```

8990 \def\newcommand{\@star@or@long\new@command}
8991 \def\new@command#1{%
8992   \@testopt{\@newcommand#1}0}
8993 \def\@newcommand#1[#2]{%
8994   \@ifnextchar [{\@xargdef#1[#2]}%
8995   {\@argdef#1[#2]}}
8996 \long\def\@argdef#1[#2]#3{%
8997   \@yargdef#1\@ne{#2}{#3}}
8998 \long\def\@xargdef#1[#2][#3]#4{%
8999   \expandafter\def\expandafter#1\expandafter{%
9000     \expandafter\@protected@testopt\expandafter #1%
9001     \csname\string#1\expandafter\endcsname{#3}}}%
9002 \expandafter\@yargdef \csname\string#1\endcsname
9003 \tw@{#2}{#4}}
9004 \long\def\@yargdef#1#2#3{%
9005   \@tempcnta#3\relax
9006   \advance \@tempcnta \@ne
9007   \let\@hash@\relax
9008   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9009   \@tempcntb #2%
9010   \@whilenum \@tempcntb < \@tempcnta
9011   \do{%
9012     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
9013     \advance\@tempcntb \@ne}%
9014   \let\@hash@##%
9015   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9016 \def\providecommand{\@star@or@long\provide@command}
9017 \def\provide@command#1{%
9018   \begingroup
9019     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9020   \endgroup
9021   \expandafter\@ifundefined\@gtempa
9022   {\def\reserved@a{\new@command#1}}%

```

```

9023     {\let\reserved@a\relax
9024     \def\reserved@a{\new@command\reserved@a}}%
9025     \reserved@a}%

9026 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9027 \def\declare@robustcommand#1{%
9028     \edef\reserved@a{\string#1}%
9029     \def\reserved@b{#1}%
9030     \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9031     \edef#1{%
9032         \ifx\reserved@a\reserved@b
9033             \noexpand\x@protect
9034             \noexpand#1%
9035         \fi
9036         \noexpand\protect
9037         \expandafter\noexpand\csname
9038             \expandafter\@gobble\string#1 \endcsname
9039     }%
9040     \expandafter\new@command\csname
9041         \expandafter\@gobble\string#1 \endcsname
9042 }
9043 \def\x@protect#1{%
9044     \ifx\protect\@typeset@protect\else
9045         \@x@protect#1%
9046     \fi
9047 }
9048 \catcode`\&=\z@ % Trick to hide conditionals
9049 \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`.

```

9050 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9051 \catcode`\&=4
9052 \ifx\in@\@undefined
9053     \def\in@#1#2{%
9054         \def\in@@##1#1##2##3\in@@{%
9055             \ifx\in@@##2\in@false\else\in@true\fi}%
9056         \in@@##2#1\in@\in@@}
9057 \else
9058     \let\bbl@tempa\@empty
9059 \fi
9060 \bbl@tempa

```

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

```

9061 \def\@ifpackagewith#1#2#3#4{#3}

```

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

```

9062 \def\@ifl@aded#1#2#3#4{}

```

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

```

9063 \ifx\@tempcnta\@undefined
9064     \csname newcount\endcsname\@tempcnta\relax
9065 \fi
9066 \ifx\@tempcntb\@undefined
9067     \csname newcount\endcsname\@tempcntb\relax
9068 \fi

```

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

```

9069 \ifx\bye\@undefined
9070   \advance\count10 by -2\relax
9071 \fi
9072 \ifx\@ifnextchar\@undefined
9073   \def\@ifnextchar#1#2#3{%
9074     \let\reserved@d=#1%
9075     \def\reserved@a{#2}\def\reserved@b{#3}%
9076     \futurelet\@let@token\@ifnch}
9077   \def\@ifnch{%
9078     \ifx\@let@token\@sptoken
9079       \let\reserved@c\@xifnch
9080     \else
9081       \ifx\@let@token\reserved@d
9082         \let\reserved@c\reserved@a
9083       \else
9084         \let\reserved@c\reserved@b
9085       \fi
9086     \fi
9087     \reserved@c}
9088   \def:{\let\@sptoken= } \: % this makes \@sptoken a space token
9089   \def:{\@xifnch} \expandafter\def: {\futurelet\@let@token\@ifnch}
9090 \fi
9091 \def\@testopt#1#2{%
9092   \@ifnextchar[#{1}{#1[#2]}}
9093 \def\@protected@testopt#1{%
9094   \ifx\protect\@typeset@protect
9095     \expandafter\@testopt
9096   \else
9097     \@x@protect#1%
9098   \fi}
9099 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9100   #2\relax}\fi}
9101 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9102   \else\expandafter\@gobble\fi{#1}}

```

## 14.4. Encoding related macros

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

```

9103 \def\DeclareTextCommand{%
9104   \@dec@text@cmd\providecommand
9105 }
9106 \def\ProvideTextCommand{%
9107   \@dec@text@cmd\providecommand
9108 }
9109 \def\DeclareTextSymbol#1#2#3{%
9110   \@dec@text@cmd\chardef#1{#2}#3\relax
9111 }
9112 \def\@dec@text@cmd#1#2#3{%
9113   \expandafter\def\expandafter#2%
9114     \expandafter{%
9115       \csname#3-cmd\expandafter\endcsname
9116       \expandafter#2%
9117       \csname#3\string#2\endcsname
9118     }%
9119   \let\@ifdefinable\@rc@ifdefinable
9120   \expandafter#1\csname#3\string#2\endcsname
9121 }
9122 \def\@current@cmd#1{%
9123   \ifx\protect\@typeset@protect\else
9124     \noexpand#1\expandafter\@gobble

```

```

9125 \fi
9126 }
9127 \def\@changed@cmd#1#2{%
9128 \ifx\protect\@typeset@protect
9129 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9130 \expandafter\ifx\csname ?\string#1\endcsname\relax
9131 \expandafter\def\csname ?\string#1\endcsname{%
9132 \@changed@x@err{#1}%
9133 }%
9134 \fi
9135 \global\expandafter\let
9136 \csname\cf@encoding\string#1\expandafter\endcsname
9137 \csname ?\string#1\endcsname
9138 \fi
9139 \csname\cf@encoding\string#1%
9140 \expandafter\endcsname
9141 \else
9142 \noexpand#1%
9143 \fi
9144 }
9145 \def\@changed@x@err#1{%
9146 \errhelp{Your command will be ignored, type <return> to proceed}%
9147 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9148 \def\DeclareTextCommandDefault#1{%
9149 \DeclareTextCommand#1?%
9150 }
9151 \def\ProvideTextCommandDefault#1{%
9152 \ProvideTextCommand#1?%
9153 }
9154 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9155 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9156 \def\DeclareTextAccent#1#2#3{%
9157 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9158 }
9159 \def\DeclareTextCompositeCommand#1#2#3#4{%
9160 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9161 \edef\reserved@b{\string##1}%
9162 \edef\reserved@c{%
9163 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9164 \ifx\reserved@b\reserved@c
9165 \expandafter\expandafter\expandafter\ifx
9166 \expandafter\@car\reserved@a\relax\relax\@nil
9167 \@text@composite
9168 \else
9169 \edef\reserved@b##1{%
9170 \def\expandafter\noexpand
9171 \csname#2\string#1\endcsname###1{%
9172 \noexpand\@text@composite
9173 \expandafter\noexpand\csname#2\string#1\endcsname
9174 ###1\noexpand\@empty\noexpand\@text@composite
9175 {##1}%
9176 }%
9177 }%
9178 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9179 \fi
9180 \expandafter\def\csname\expandafter\string\csname
9181 #2\endcsname\string#1-\string#3\endcsname{#4}
9182 \else
9183 \errhelp{Your command will be ignored, type <return> to proceed}%
9184 \errmessage{\string\DeclareTextCompositeCommand\space used on
9185 inappropriate command \protect#1}
9186 \fi
9187 }

```

```

9188 \def\@text@composite#1#2#3\@text@composite{%
9189   \expandafter\@text@composite@x
9190   \csname\string#1-\string#2\endcsname
9191 }
9192 \def\@text@composite@x#1#2{%
9193   \ifx#1\relax
9194     #2%
9195   \else
9196     #1%
9197   \fi
9198 }
9199 %
9200 \def\@strip@args#1:#2-#3\@strip@args{#2}
9201 \def\DeclareTextComposite#1#2#3#4{%
9202   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9203   \bgroup
9204     \lccode`\@=#4%
9205     \lowercase{%
9206       \egroup
9207       \reserved@a @%
9208     }%
9209 }
9210 %
9211 \def\UseTextSymbol#1#2{#2}
9212 \def\UseTextAccent#1#2#3{}
9213 \def\@use@text@encoding#1{}
9214 \def\DeclareTextSymbolDefault#1#2{%
9215   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9216 }
9217 \def\DeclareTextAccentDefault#1#2{%
9218   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9219 }
9220 \def\cf@encoding{OT1}

```

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

```

9221 \DeclareTextAccent{"}{OT1}{127}
9222 \DeclareTextAccent{'}{OT1}{19}
9223 \DeclareTextAccent{^}{OT1}{94}
9224 \DeclareTextAccent{\`}{OT1}{18}
9225 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9226 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9227 \DeclareTextSymbol{\textquotedblright}{OT1}{'\'}
9228 \DeclareTextSymbol{\textquoteleft}{OT1}{'\'}
9229 \DeclareTextSymbol{\textquoteright}{OT1}{'\'}
9230 \DeclareTextSymbol{\i}{OT1}{16}
9231 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9232 \ifx\scriptsize\undefined
9233   \let\scriptsize\sevenrm
9234 \fi

```

And a few more “dummy” definitions.

```

9235 \def\language{english}%
9236 \let\bbl@opt@shorthands\@nnil
9237 \def\bbl@ifshorthand#1#2#3#2{%
9238   \let\bbl@language@opts\@empty
9239   \let\bbl@provide@locale\relax
9240   \ifx\babeloptionstrings\undefined
9241     \let\bbl@opt@strings\@nnil

```



```

9242 \else
9243 \let\bbl@opt@strings\babeloptionstrings
9244 \fi
9245 \def\BabelStringsDefault{generic}
9246 \def\bbl@tempa{normal}
9247 \ifx\babeloptionmath\bbl@tempa
9248 \def\bbl@mathnormal{\noexpand\textormath}
9249 \fi
9250 \def\AfterBabelLanguage#1#2{}
9251 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9252 \let\bbl@afterlang\relax
9253 \def\bbl@opt@safe{BR}
9254 \ifx\@uclclist\undefined\let\@uclclist\empty\fi
9255 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9256 \expandafter\newif\csname ifbbl@single\endcsname
9257 \chardef\bbl@bidimode\z@
9258 <</Emulate LaTeX>>

A proxy file:
9259 <*plain>
9260 \input babel.def
9261 </plain>

```

## 15. Acknowledgements

In the initial stages of the development of babel, Bernd Raichle provided many helpful suggestions and Michel Goossens supplied contributions for many languages. Ideas from Nico Poppelier, Piet van Oostrum and many others have been used. Paul Wackers and Werenfried Spit helped find and repair bugs.

More recently, there are significant contributions by Salim Bou, Ulrike Fischer, Loren Davis and Udi Fogiel.

Barbara Beeton has helped in improving the manual.

There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\TeX$  styles*, *TUGboat* 10 (1989) #3, pp. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\TeX$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018.
- [10] Hubert Partl, *German  $\TeX$* , *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International  $\TeX$  is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\TeX$* , Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).