

Babel

Code

Version 26.15
2026/10/29

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

LuaT_EX

pdfT_EX

XeT_EX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
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	12
3.6	Selecting the language	15
3.7	Errors	23
3.8	More on selection	24
3.9	Short tags	25
3.10	Compatibility with language.def	26
3.11	Hooks	26
3.12	Setting up language files	27
3.13	Shorthands	29
3.14	Language attributes	38
3.15	Support for saving and redefining macros	40
3.16	French spacing	41
3.17	Hyphens	42
3.18	Multiencoding strings	44
3.19	Tailor captions	48
3.20	Making glyphs available	49
3.20.1	Quotation marks	49
3.20.2	Letters	51
3.20.3	Shorthands for quotation marks	52
3.20.4	Umlauts and tremas	52
3.21	Layout	54
3.22	Load engine specific macros	54
3.23	Creating and modifying languages	54
3.24	Main loop in ‘provide’	62
3.25	Processing keys in ini	67
3.26	French spacing (again)	73
3.27	Handle language system	74
3.28	Numerals	75
3.29	Casing	76
3.30	Getting info	77
3.31	BCP 47 related commands	78
4	Adjusting the Babel behavior	79
4.1	Cross referencing macros	81
4.2	Layout	84
4.3	Marks	86
4.4	Other packages	87
4.4.1	ifthen	87
4.4.2	varioref	87
4.4.3	hhline	88
4.5	Encoding and fonts	89
4.6	Basic bidi support	90
4.7	Local Language Configuration	93
4.8	Language options	94
5	Tagging	98
6	Error messages	99

7 Loading hyphenation patterns	103
8 luatex + xetex: common stuff	107
9 Hooks for XeTeX and LuaTeX	111
9.1 XeTeX	111
9.2 Support for interchar	112
9.3 Layout	114
9.4 8-bit TeX	116
9.5 LuaTeX	117
9.6 Southeast Asian scripts	124
9.7 CJK line breaking	125
9.8 Arabic justification	127
9.9 Common stuff	131
9.10 Automatic fonts and ids switching	132
9.11 Bidi	139
9.12 Layout	141
9.13 Lua: transforms	150
9.14 Lua: Auto bidi with basic and basic-r	161
10 Data for CJK	174
11 The ‘nil’ language	174
12 Calendars	175
12.1 Islamic	176
12.2 Hebrew	177
12.3 Persian	181
12.4 Coptic and Ethiopic	182
12.5 Julian	182
12.6 Buddhist	183
13 Acknowledgements	184

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

1. Identification and loading of required files

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain and, in part, by the base option

switch.def is not used. Kept for compatibility.

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=26.15>>
2 <<date=2026/10/29>>
```

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

```
3 <<*Basic macros>>
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8   {\def#1{#2}}%
9   {\expandafter\def\expandafter#1\expandafter{#1#2}}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@carg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
16 \def\bbl@carg#1{\csname bbl@#1\endcsname}
17 \def\bbl@cl#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-statement¹. 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}}

```

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

\bbl@ifunset It does the same as \ifundefined. Formerly defined for ϵ -tex engines based on \ifcsname, which is more efficient, and does not waste memory. Now \ifundefined is based on \ifcsname, too.

```
56 \let\bbl@ifunset\ifundefined
```

\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,

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

```
62 \def\bbl@forkv#1#2{%
63   \def\bbl@kvcmd##1##2##3{#2}%
64   \bbl@kvnxt#1,\@nil,}
65 \def\bbl@kvnxt#1,{%
66   \ifx\@nil#1\relax\else
67     \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
68     \expandafter\bbl@kvnxt
69   \fi}
70 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
71   \bbl@trim\def\bbl@forkv@a{#1}%
72   \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).

```
73 \def\bbl@vforeach#1#2{%
74   \def\bbl@forcmd##1{#2}%
75   \bbl@fornext#1,\@nil,}
76 \def\bbl@fornext#1,{%
77   \ifx\@nil#1\relax\else
78     \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
79     \expandafter\bbl@fornext
80   \fi}
81 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

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

```
82 \global\let\bbl@done\empty
83 \def\bbl@once#1#2{%
84   \bbl@xin@{, #1,}{,\bbl@done,}%
85   \ifin\else
86     #2%
87   \xdef\bbl@done{\bbl@done, #1,}%
88   \fi}
89 % \end{macrocode}
90 %
91 % \macro{\bbl@replace}
92 %
93 % Returns implicitly |\toks@| with the modified string.
94 %
95 % \begin{macrocode}
96 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
97   \toks@{}}%
98 \def\bbl@replace@aux##1#2##2#2{%
99   \ifx\bbl@nil##2%
100     \toks@\expandafter{\the\toks@##1}%
101   \else
102     \toks@\expandafter{\the\toks@##1#3}%
103   \fi}
```

```

103 \bbl@afterfi
104 \bbl@replace@aux##2#2%
105 \fi}%
106 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
107 \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).

```

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

```

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

```

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

```

```

156 \@ne
157 \fi

```

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

```

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

```

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

```

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

```

Ordered key/value lists (aka associative arrays). The setter either replaces the current value if the key exists, or adds the new key/value at the end. The getter returns either the value if the key exists or \relax. The code is based on arguments delimited by the keys.

```

177 \def\bbl@empty{\@empty}
178 \def\bbl@array@set#1#2#3{%
179 \def\bbl@array@aux##1\bbl@elt#2\@##2##3\bbl@array@aux##4{%
180 \ifx\bbl@noelt##2%
181 \def#1{##1\bbl@elt#2\@##4}}%
182 \else
183 \def\bbl@array@aux####1\bbl@elt#2\@####2###3\bbl@array@aux####4{%
184 \def#1{####1\bbl@elt#2\@####4}###3}}%
185 \expandafter\bbl@array@aux#1\bbl@array@aux{#3}%
186 \fi}%
187 \expandafter\bbl@array@aux#1\bbl@elt#2\@##2\bbl@noelt\bbl@array@aux{#3}}
188 \def\bbl@array@get#1#2#3{%
189 \def\bbl@array@aux##1\bbl@elt#2\@##2##3\bbl@array@aux{%
190 \ifx\bbl@empty##2\@empty\let#3\relax\else\def#3{##2}\fi}%
191 \expandafter\bbl@array@aux#1\bbl@elt#2\@##2\bbl@empty\bbl@array@aux}

```

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).

```

192 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
193 \toks@{\expandafter\expandafter\expandafter{%
194 \cename extras\languagename\endcsname}%
195 \bbl@exp{\in{#1}}{\the\toks@}}%
196 \ifin@\else
197 \@temptokena{#2}%
198 \edef\bbl@tempc{\the\@temptokena\the\toks@}%
199 \toks@\expandafter{\bbl@tempc#3}%
200 \expandafter\edef\cename extras\languagename\endcsname{\the\toks@}%
201 \fi}
202 <</Basic macros>>

```

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

```

203 <<(*Make sure ProvidesFile is defined)>> ▶

```

```

204 \ifx\ProvidesFile\@undefined
205 \def\ProvidesFile#1[#2 #3 #4]{%
206 \wlog{File: #1 #4 #3 <#2>}%
207 \let\ProvidesFile\@undefined}
208 \fi
209 <(/Make sure ProvidesFile is defined)>

```

3.1. A few core definitions

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

```

210 <(*Define core switching macros)>
211 \ifx\language\@undefined
212 \csname newcount\endcsname\language
213 \fi
214 <(/Define core switching macros)>

```

\last@language Another counter is used to keep track of the allocated languages. T_EX and L^AT_EX reserves for this purpose the count 19.

\addlanguage This macro was introduced for T_EX < 2. Preserved for compatibility.

```

215 <(*Define core switching macros)>
216 \countdef\last@language=19
217 \def\addlanguage{\csname newlanguage\endcsname}
218 <(/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. L^AT_EX: babel.sty (start)

Here starts the style file for L^AT_EX. It also takes care of a number of compatibility issues with other packages.

```

219 <(*package)>
220 \NeedsTeXFormat{LaTeX2e}
221 \ProvidesPackage{babel}%
222 [<@date> v<@version>]
223 The multilingual framework for LuaLaTeX, pdfLaTeX 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).

```

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

```

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

```

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

```

261 <@Basic macros@>
262 \ifpackagewith{babel}{silent}
263   {\let\bbl@info\@gobble
264    \let\bbl@infowarn\@gobble
265    \let\bbl@warning\@gobble}
266 {}
267 %
268 \def\AfterBabelLanguage#1{%
269   \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.

```

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

```

3.3. base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets ver@babel.sty so that L^AT_EX 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.

```

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

```

308 \bbl@trace{key=value and another general options}
309 \bbl@csarg\let\tempa\expandafter\csname opt@babel.sty\endcsname
310 \def\bbl@tempb#1.#2{% Removes trailing dot
311   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
312 \def\bbl@tempe#1=#2\@@{%
313   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
314 \def\bbl@tempd#1.#2\@nnil{%
315   \ifx\@empty#2%
316     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
317   \else
318     \in@{,provide=}{, #1}%
319     \ifin@
320       \edef\bbl@tempc{%
321         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
322     \else
323       \in@{$modifiers$}{$#1$}%
324       \ifin@
325         \bbl@tempe#2\@@
326       \else
327         \in@{=}{#1}%
328         \ifin@
329           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
330         \else
331           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
332           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
333         \fi
334       \fi
335     \fi
336   \fi}
337 \let\bbl@tempc\@empty
338 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
339 \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.

```

340 \DeclareOption{KeepShorthandsActive}{}
341 \DeclareOption{activeacute}{}
342 \DeclareOption{activegrave}{}
343 \DeclareOption{debug}{}
344 \DeclareOption{debug-german}{} % Temporary
345 \DeclareOption{noconfigs}{}
346 \DeclareOption{showlanguages}{}
347 \DeclareOption{silent}{}
348 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
349 \chardef\bbl@iniflag\z@
350 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
351 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
352 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
353 \chardef\bbl@ldfflag\z@
354 \DeclareOption{provide=!}{\chardef\bbl@ldfflag\@ne} % main = 1
355 \DeclareOption{provide+=!}{\chardef\bbl@ldfflag\tw@} % second = 2
356 \DeclareOption{provide*=!}{\chardef\bbl@ldfflag\thr@@} % second + main
357 \DeclareOption{licr=extended}{}
358 \DeclareOption{licr=unextended}{}
359 \let\bbl@ifmetadata\@gobble
360 \DeclareOption{babel-metadata=on}{%
361   \IfDocumentMetadataT{\AtEndOfPackage{\input{tagbabel.def}}}}
362 \DeclareOption{babel-metadata=tagging}{%
363   \IfDocumentMetadataT{\AtEndOfPackage{\input{tagbabel.def}}}}
364 % Don't use. Experimental.
365 \newif\ifbbl@single
366 \DeclareOption{selectors=off}{\bbl@singletrue}
367 <@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.

```

368 \let\bbl@opt@shorthands\@nnil
369 \let\bbl@opt@config\@nnil
370 \let\bbl@opt@main\@nnil
371 \let\bbl@opt@headfoot\@nnil
372 \let\bbl@opt@layout\@nnil
373 \let\bbl@opt@provide\@nnil

```

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

```

374 \def\bbl@tempa#1=#2\bbl@tempa{%
375   \bbl@csarg\ifx{opt#1}\@nnil
376     \bbl@csarg\edef{opt#1}{#2}%
377   \else
378     \bbl@error{bad-package-option}{#1}{#2}{}%
379   \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.

```

380 \let\bbl@language@opts\@empty
381 \DeclareOption*{%
382   \bbl@xin@{\string=}{\CurrentOption}%
383   \ifin@
384     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
385   \else
386     \bbl@add@list\bbl@language@opts{\CurrentOption}%
387   \fi}

```

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

```

388 \ProcessOptions*

```

3.5. Post-process some options

```
389 \ifx\bbl@opt@provide\@nnil
390 \let\bbl@opt@provide\@empty %%%% MOVE above
391 \else
392 \chardef\bbl@iniflag\@ne
393 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
394 \in@{,provide,}{, #1,}%
395 \ifin@
396 \def\bbl@opt@provide{#2}%
397 \fi}
398 \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=...`

```
399 \bbl@trace{Conditional loading of shorthands}
400 \def\bbl@sh@string#1{%
401 \ifx#1\@empty\else
402 \ifx#1t\string~%
403 \else\ifx#1c\string,%
404 \else\string#1%
405 \fi\fi
406 \expandafter\bbl@sh@string
407 \fi}
408 \ifx\bbl@opt@shorthands\@nnil
409 \def\bbl@ifshorthand#1#2#3{#2}%
410 \else\ifx\bbl@opt@shorthands\@empty
411 \def\bbl@ifshorthand#1#2#3{#3}%
412 \else
```

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

```
413 \def\bbl@ifshorthand#1{%
414 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
415 \ifin@
416 \expandafter\@firstoftwo
417 \else
418 \expandafter\@secondoftwo
419 \fi}
```

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

```
420 \edef\bbl@opt@shorthands{%
421 \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.

```
422 \bbl@ifshorthand{'}%
423 {\PassOptionsToPackage{activeacute}{babel}}{}
424 \bbl@ifshorthand{`}%
425 {\PassOptionsToPackage{activegrave}{babel}}{}
426 \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.

```
427 \ifx\bbl@opt@headfoot\@nnil\else
428 \g@addto@macro\@resetactivechars{%
429 \set@typeset@protect
430 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
431 \let\protect\noexpand}
432 \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.

```
433 \ifx\bbl@opt@safe\@undefined
```

```

434 \def\bbl@opt@safe{BR}
435 % \let\bbl@opt@safe\@empty % Pending of \cite
436 \fi

```

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

```

437 \bbl@trace{Defining IfBabelLayout}
438 \ifx\bbl@opt@layout\@nnil
439 \newcommand\IfBabelLayout[3]{#3}%
440 \else
441 \bbl@exp{\bbl@forkv{\@nameuse{\raw@opt@babel.sty}}}{%
442 \in{,layout,}{, #1,}%
443 \ifin@
444 \def\bbl@opt@layout{#2}%
445 \bbl@replace\bbl@opt@layout{ }{.}%
446 \fi}
447 \newcommand\IfBabelLayout[1]{%
448 \@expandtwoargs\in{.#1.}{.\bbl@opt@layout.}%
449 \ifin@
450 \expandafter\@firstoftwo
451 \else
452 \expandafter\@secondoftwo
453 \fi}
454 \fi

455 \def\bbl@version{<@version@>}
456 \def\bbl@date{<@date@>}
457 <@Define core switching macros@>

```

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

```

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

```

473 \def\bbl@fixname#1{%
474 \begingroup
475 \def\bbl@tempe{l@}%
476 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
477 \bbl@tempd
478 {\lowercase\expandafter{\bbl@tempd}%
479 {\uppercase\expandafter{\bbl@tempd}%
480 \@empty
481 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
482 \uppercase\expandafter{\bbl@tempd}}}%
483 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
484 \lowercase\expandafter{\bbl@tempd}}}%

```

```

485 \empty
486 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
487 \bbl@tempd
488 \bbl@expf{\bbl@usehooks{language}{\language}{#1}}%
489 \def\bbl@iflanguage#1{%
490 \ifundefined{l@#1}{\no!anerr{#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, casing is the usual one, so that sr-latn-ba becomes fr-Latn-BA.

\bbl@bcpllookup returns \bbl@bcp with either the found ini tag or \relax. Temporary version, to be refactored with the new \text_bcp_parse:n, which is not currently fully expandable with invalid BCP 47 tags. The argument may be in some cases either a language name or a tag (e.g., \foreignlanguage. This was a huge mistake, because of the potential ambiguities (for example, the name vai-latin is also a syntactically valid tag).

```

491 \def\bbl@cntchars#1{\bbl@cntchars@i#1876543210\@{}}
492 \def\bbl@cntchars@i#1#2#3#4#5#6#7#8#9\@{\@car#9\@nil}%
493 \def\bbl@bcpllookup#1{%
494 \let\bbl@templ\empty % language
495 \let\bbl@temps\empty % script
496 \let\bbl@tempr\empty % regions
497 \let\bbl@tempv\empty % variant
498 \edef\bbl@tempa{#1}%
499 \bbl@replace\bbl@tempa{-}{,}%
500 \bbl@replace\bbl@tempa_{,}%
501 \bbl@foreach\bbl@tempa{%
502 \ifx\bbl@templ\empty
503 \lowercase{\def\bbl@templ{##1}}%
504 \else\ifnum\bbl@cntchars{##1}=4
505 \bbl@xin@{\@car##1\@nil}{0123456789}%
506 \ifin@
507 \def\bbl@tempv{##1}% eg, 1901
508 \else
509 \uppercase{\edef\bbl@tempb{\@car##1\@nil}}%
510 \lowercase{\edef\bbl@tempb{\bbl@tempb\@gobble##1}}%
511 \fi
512 \else\ifnum\bbl@cntchars{##1}<4
513 \uppercase{\def\bbl@tempr{##1}}%
514 \else\ifnum\bbl@cntchars{##1}>4
515 \lowercase{\def\bbl@tempv{##1}}%
516 \fi\fi\fi\fi}%
517 \bbl@expf{\bbl@bcpllookup@i{\bbl@templ}{\bbl@temps}{\bbl@tempr}{\bbl@tempv}}%
518 \def\bbl@bcpllookup@try#1{%
519 \ifx\bbl@bcp\relax
520 \IfFileExists{babel-#1\bbl@tempe.ini}{\edef\bbl@bcp{#1\bbl@tempe}}{%
521 \fi}
522 \def\bbl@bcpllookup@i#1#2#3#4{%
523 \let\bbl@bcp\relax
524 \def\bbl@tempe{#4}%
525 \ifx\bbl@tempe\empty\else
526 \edef\bbl@tempe{-#4}%
527 \fi
528 \edef\bbl@tempa{#2#3\bbl@tempe}% en
529 \ifx\bbl@tempa\empty
530 \bbl@bcpllookup@try{#1}%
531 \else
532 \edef\bbl@tempa{#3\bbl@tempe}% en-Latn
533 \ifx\bbl@tempa\empty
534 \bbl@bcpllookup@try{#1-#2}%
535 \bbl@bcpllookup@try{#1}%
536 \else
537 \edef\bbl@tempa{#2\bbl@tempe}% en-US, en-001

```

```

538     \ifx\bbl@tempa\@empty
539       \bbl@bcpllookup@try{#1-#3}%
540       \bbl@bcpllookup@try{#1}%
541     \else % en-Latn-US
542       \bbl@bcpllookup@try{#1-#2-#3}%
543       \bbl@bcpllookup@try{#1-#2}%
544       \bbl@bcpllookup@try{#1-#3}%
545       \bbl@bcpllookup@try{#1}%
546     \fi
547   \fi
548 \fi
549 \ifx\bbl@bcp\relax
550   \ifx\bbl@tempe\@empty\else\bbl@bcpllookup@i{#1}{#2}{#3}{ }\fi%
551 \fi}
552 %
553 \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.

```

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

```

3.6. 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.

```

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

```

565 \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.

```

566 \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.

```

567 \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:

```

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

```

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

```

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

```

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

```

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

```

606 \let\bbl@select@opts\@empty
607 \expandafter\def\csname selectlanguage \endcsname{%
608   \ifnextchar[\bbl@select@s{\bbl@select@s[]}}
609 \def\bbl@select@s[#1]#2{%
610   \bbl@forkv{#1}{\bbl@array@set\bbl@select@opts{##1}{##2}}%
611   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
612   \bbl@push@language
613   \aftergroup\bbl@pop@language
614   \bbl@set@language{#2}}
615 \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).

```

616 \def\BabelContentsFiles{toc,lof,lot}
617 \def\bbl@set@language#1{% from selectlanguage, pop@
618   % The old buggy way. Preserved for compatibility, but simplified
619   \edef\language{\expandafter\string#1\@empty}%
620   \select@language{\language}%
621   \bbl@array@get\bbl@select@opts{main}\bbl@tempa
622   \ifx\bbl@tempa\relax\else
623     \let\bbl@main@language\localename
624     \let\mainlocalename\localename
625   \fi
626   % write to aux files
627   \expandafter\ifx\csname date\language\endcsname\relax\else
628     \if@filesw
629       \bbl@array@get\bbl@select@opts{nofiles}\bbl@tempa
630       \ifx\bbl@tempa\relax
631         \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
632           \bbl@savelastskip
633           \protected@write\auxout{}\string\babel@aux{\bbl@auxname}{}}%
634           \bbl@restorelastskip
635         \fi
636         \bbl@usehooks{write}{}%
637       \fi
638     \fi
639   \fi
640   \let\bbl@select@opts\@empty}
641 %
642 \let\bbl@restorelastskip\relax
643 \let\bbl@savelastskip\relax
644 %
645 \def\select@language#1{% from set@, babel@aux, babel@toc
646   \ifx\bbl@select@name\@empty
647     \def\bbl@select@name{select}%
648   \fi
649   % set hmap
650   \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
651   % set name (when coming from babel@aux)
652   \edef\language{#1}%

```

```

653 \bbl@fixname\language\language
654 % define \localename when coming from set@, with a trick
655 \ifx\scantokens\undefined
656 \def\localename{??}%
657 \else
658 \bbl@exp{\scantokens{\def\localename{\language\language}\noexpand}\relax}%
659 \fi
660 \bbl@provide@locale
661 \bbl@iflanguage\language\language{%
662 \let\bbl@select@type\z@
663 \expandafter\bbl@switch\expandafter{\language}}
664 \def\babel@aux#1#2{%
665 \select@language{#1}%
666 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
667 \@writefile{#1}{\babel@toc{#1}{#2}\relax}}%
668 \def\babel@toc#1#2{%
669 \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`.

```

670 \newif\ifbbl@usedategroup
671 \let\bbl@savextras\empty
672 \def\bbl@switch#1{% from select@, foreign@
673 % restore
674 \originalTeX
675 \expandafter\def\expandafter\originalTeX\expandafter{%
676 \csname noextras#1\endcsname
677 \let\originalTeX\empty
678 \babel@beginsave}%
679 \bbl@usehooks{afterreset}{}%
680 \languageshorthands{none}%
681 % set the locale id
682 \bbl@id@assign
683 % switch captions, date
684 \bbl@bsphack
685 \ifcase\bbl@select@type
686 \csname captions#1\endcsname\relax
687 \csname date#1\endcsname\relax
688 \else
689 \bbl@array@get\bbl@select@opts{captions}\bbl@tempa
690 \ifx\bbl@tempa\relax\else
691 \csname captions#1\endcsname\relax
692 \fi
693 \bbl@array@get\bbl@select@opts{date}\bbl@tempa
694 \ifx\bbl@tempa\relax\else % if \foreign... within \<language>date
695 \csname date#1\endcsname\relax
696 \fi
697 \fi
698 \bbl@esphack
699 % switch extras
700 \csname bbl@preextras@#1\endcsname

```

```

701 \bbl@usehooks{beforeextras}{}%
702 \csname extras#1\endcsname\relax
703 \bbl@usehooks{afterextras}{}%
704 % > babel-ensure
705 % > babel-sh-<short>
706 % > babel-bidi
707 % > babel-fontspec
708 \let\bbl@savextras\@empty
709 % hyphenation - case mapping
710 \ifcase\bbl@opt@hyphenmap\or
711 \def\BabelLower##1##2{\lccode##1=##2\relax}%
712 \ifnum\bbl@hymap>4\else
713 \csname\language @bbl@hyphenmap\endcsname
714 \fi
715 \chardef\bbl@opt@hyphenmap\z@
716 \else
717 \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
718 \csname\language @bbl@hyphenmap\endcsname
719 \fi
720 \fi
721 \let\bbl@hymap\@cclv
722 % hyphenation - select rules
723 \ifnum\csname l@\language\endcsname=\l@unhyphenated
724 \edef\bbl@tempa{u}%
725 \else
726 \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
727 \fi
728 % linebreaking - handle u, e, k (v in the future)
729 \bbl@xin@{/u}{/\bbl@tempa}%
730 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
731 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
732 \ifin@else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (e.g., Tibetan)
733 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
734 % hyphenation - save mins
735 \babel@savevariable\lefthyphenmin
736 \babel@savevariable\righthyphenmin
737 \ifnum\bbl@engine=\@ne
738 \babel@savevariable\hyphenationmin
739 \fi
740 \ifin@
741 % unhyphenated/kashida/elongated/padding = allow stretching
742 \language\l@unhyphenated
743 \babel@savevariable\emergencystretch
744 \emergencystretch\maxdimen
745 \babel@savevariable\hbadness
746 \hbadness\@M
747 \else
748 % other = select patterns
749 \bbl@patterns{#1}%
750 \fi
751 % hyphenation - set mins
752 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
753 \set@hyphenmins\tw@thr@@\relax
754 \@nameuse{\bbl@hyphenmins}%
755 \else
756 \expandafter\expandafter\expandafter\set@hyphenmins
757 \csname #1hyphenmins\endcsname\relax
758 \fi
759 \@nameuse{\bbl@hyphenmins}%
760 \@nameuse{\bbl@hyphenmins@\language}%
761 \@nameuse{\bbl@hyphenatmin}%
762 \@nameuse{\bbl@hyphenatmin@\language}%
763 \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.

```

764 \edef\otherlanguage{%
765   \noexpand\protect
766   \expandafter\noexpand\csname otherlanguage \endcsname}
767 \expandafter\def\csname otherlanguage \endcsname{%
768   \@ifstar{\@nameuse{otherlanguage*}}\bbl@otherlanguage}
769 \def\bbl@otherlanguage#1{%
770   \def\bbl@selectorname{other}%
771   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
772   \csname selectlanguage \endcsname{#1}%
773   \ignorespaces}

```

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

```

774 \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`.

```

775 \expandafter\def\csname otherlanguage*\endcsname{%
776   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
777 \def\bbl@otherlanguage@s[#1]#2{%
778   \def\bbl@selectorname{other*}%
779   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
780   \bbl@forkv{#1}{\bbl@array@set\bbl@select@opts{##1}{##2}}%
781   \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”.

```

782 \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`.

```

783 \providecommand\bbl@beforeforeign{}
784 \edef\foreignlanguage{%
785   \noexpand\protect
786   \expandafter\noexpand\csname foreignlanguage \endcsname}
787 \expandafter\def\csname foreignlanguage \endcsname{%
788   \@ifstar\bbl@foreign@s\bbl@foreign@x}
789 \providecommand\bbl@foreign@x[3][]{%
790   \begingroup
791   \def\bbl@selectorname{foreign}%

```

```

792 \bbl@forkv{#1}{\bbl@array@set\bbl@select@opts{##1}{##2}}%
793 \let\BabelText\@firstofone
794 \bbl@beforeforeign
795 \foreign@language{#2}%
796 \bbl@ifmetadata{\UseTaggingSocket{babel/foreign/begin}}%
797 \bbl@usehooks{foreign}{}%
798 \BabelText{#3}% Now in horizontal mode!
799 \bbl@ifmetadata{\UseTaggingSocket{babel/foreign/end}}%
800 \endgroup}
801 \def\bbl@foreign@s#1#2{%
802 \begingroup
803 {\par}%
804 \def\bbl@select@name{foreign*}%
805 \let\bbl@select@opts\@empty
806 \let\BabelText\@firstofone
807 \foreign@language{#1}%
808 \bbl@usehooks{foreign*}{}%
809 \bbl@dirparastext
810 \BabelText{#2}% Still in vertical mode!
811 {\par}%
812 \endgroup}
813 \providecommand\BabelWrapText[1]{%
814 \def\bbl@tempa{\def\BabelText###1}%
815 \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.

```

816 \def\foreign@language#1{%
817 % set name
818 \edef\languagename{#1}%
819 \ifbbl@usedategroup
820 \bbl@array@set\bbl@select@opts{date}{\@empty}%
821 \bbl@usedategroupfalse
822 \fi
823 \bbl@fixname\languagename
824 \let\localename\languagename
825 \bbl@provide@locale
826 \bbl@iflanguage\languagename{%
827 \let\bbl@select@type\@ne
828 \expandafter\bbl@switch\expandafter{\languagename}}}

```

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

```

829 \let\bbl@select@name\@empty
830 \def\IfBabelSelectorTF#1{%
831 \bbl@xin{,\bbl@select@name,}{,\zap@space#1 \@empty,}%
832 \ifin@
833 \expandafter\@firstoftwo
834 \else
835 \expandafter\@secondoftwo
836 \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.

```

837 \let\bbl@hyphlist\@empty
838 \let\bbl@hyphenation@\relax

```

```

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

```

hyphenrules It can be used to select *just* the hyphenation rules. It does *not* change \languagename 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 other\language*.

```

865 \def\hyphenrules#1{%
866   \edef\bbl@tempf{#1}%
867   \bbl@fixname\bbl@tempf
868   \bbl@iflanguage\bbl@tempf{%
869     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
870     \ifx\languageshorthands\@undefined\else
871       \languageshorthands{none}%
872     \fi
873     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
874       \set@hyphenmins\tw@\thr@@\relax
875     \else
876       \expandafter\expandafter\expandafter\set@hyphenmins
877       \csname\bbl@tempf hyphenmins\endcsname\relax
878     \fi}}
879 \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.

```

880 \def\providehyphenmins#1#2{%
881   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
882     \@namedef{#1hyphenmins}{#2}%
883   \fi}

```

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

```

884 \def\set@hyphenmins#1#2{%
885   \lefthyphenmin#1\relax
886   \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.

```

887 \ifx\ProvidesFile\@undefined
888   \def\ProvidesLanguage#1[#2 #3 #4]{%
889     \wlog{Language: #1 #4 #3 <#2>}%
890   }
891 \else
892   \def\ProvidesLanguage#1{%
893     \begingroup
894       \catcode`\ 10 %
895       \@makeother\/%
896       \@ifnextchar[%]
897         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
898   \def\@provideslanguage#1[#2]{%
899     \wlog{Language: #1 #2}%
900     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
901   \endgroup}
902 \fi

```

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

```

903 \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.

```

904 \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’:

```

905 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
906 \let\uselocale\setlocale
907 \let\locale\setlocale
908 \let\selectlocale\setlocale
909 \let\textlocale\setlocale
910 \let\textlanguage\setlocale
911 \let\languagegettext\setlocale

```

3.7. 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 $\text{\LaTeX 2}_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

```

912 \edef\bbl@nulllanguage{\string\language=0}
913 \def\bbl@nocaption#1#2{\NoCaseChange{\protect\bbl@nocaption@i{#1}{#2}}}
914 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
915   \global\@namedef{#2}{\textbf{?#1?}}%
916   \@nameuse{#2}%
917   \edef\bbl@tempa{#1}%
918   \bbl@replace\bbl@tempa{name}}}%
919 \bbl@warning{%
920   \@backslashchar#1 not set for '\language'. Please,\%

```

```

921     define it after the language has been loaded\\%
922     (typically in the preamble) with:\\%
923     \string\setlocalecaption{\language\name}{\bbl@tempa}{.}\\%
924     Feel free to contribute on github.com/latex3/babel.\\%
925     Reported}}
926 \def\bbl@tentative{\protect\bbl@tentative@i}
927 \def\bbl@tentative@i#1{%
928   \bbl@warning{%
929     Some functions for '#1' are tentative.\\%
930     They might not work as expected and their behavior\\%
931     could change in the future.\\%
932     Reported}}
933 \def\@nolannerr#1{\bbl@error{undefined-language}{#1}{}}
934 \def\@nopatterns#1{%
935   \bbl@warning
936     {No hyphenation patterns were preloaded for\\%
937     the language '#1' into the format.\\%
938     Please, configure your TeX system to add them and\\%
939     rebuild the format. Now I will use the patterns\\%
940     preloaded for \bbl@nulllanguage\space instead}}
941 \let\bbl@usehooks\@gobbletwo

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

942 \ifx\bbl@onlyswitch\@empty\endinput\fi

```

3.8. 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.

```

943 \bbl@trace{Defining babelensure}
944 \newcommand\babelensure[2][]{%
945   \AddBabelHook{babel-ensure}{afterextras}{%
946     \ifcase\bbl@select@type
947       \bbl@ccl{e}%
948       \fi}%
949   \begingroup
950     \let\bbl@ens@include\@empty
951     \let\bbl@ens@exclude\@empty
952     \def\bbl@ens@fontenc{\relax}%
953     \def\bbl@tempb##1{%
954       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
955     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
956     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
957     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
958     \def\bbl@tempc{\bbl@ensure}%
959     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
960       \expandafter{\bbl@ens@include}}%
961     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
962       \expandafter{\bbl@ens@exclude}}%
963     \toks@\expandafter{\bbl@tempc}%
964     \bbl@exp{%
965       \endgroup
966       \def<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
967 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
968   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list

```

```

969 \ifx##1\@undefined % 3.32 - Don't assume the macro exists
970 \edef##1{\noexpand\bb1@nocaption
971 { \bb1@stripslash##1}{\language\bb1@stripslash##1}}%
972 \fi
973 \ifx##1\@empty\else
974 \in@{##1}{#2}%
975 \ifin@ \else
976 \let\bb1@tempc\language
977 \bb1@replace\bb1@tempc{-}{@}%
978 \bb1@ifunset{\bb1@ensure@\bb1@tempc}%
979 {\bb1@exp{%
980 \\\DeclareRobustCommand\<\bb1@ensure@\bb1@tempc>[1]{%
981 \\\foreignlanguage{\language}%
982 {\ifx\relax#3\else
983 \\\fontencoding{#3}\selectfont
984 \fi
985 #####1}}}%
986 }%
987 \toks@ \expandafter{##1}%
988 \edef##1{%
989 \bb1@csarg\noexpand\ensure@\bb1@tempc}%
990 {\the\toks@}}%
991 \fi
992 \expandafter\bb1@tempb
993 \fi}%
994 \expandafter\bb1@tempb\bb1@captionslist\today\@empty
995 \def\bb1@tempa##1{% elt for include list
996 \ifx##1\@empty\else
997 \let\bb1@tempc\language
998 \bb1@replace\bb1@tempc{-}{@}%
999 \bb1@csarg\in@\ensure@\bb1@tempc\expandafter\expandafter{##1}%
1000 \ifin@ \else
1001 \bb1@tempb##1\@empty
1002 \fi
1003 \expandafter\bb1@tempa
1004 \fi}%
1005 \bb1@tempa##1\@empty}
1006 \def\bb1@captionslist{%
1007 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1008 \contentsname\listfigurename\listtablename\indexname\figurename
1009 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1010 \alsoname\proofname\glossaryname}

```

3.9. 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.

```

1011 \bb1@trace{Short tags}
1012 \newcommand\babeltags[1]{%
1013 \edef\bb1@tempa{\zap@space#1 \@empty}%
1014 \def\bb1@tempb##1=##2\@{#%
1015 \edef\bb1@tempc{%
1016 \noexpand\newcommand
1017 \expandafter\noexpand\csname ##1\endcsname{%
1018 \noexpand\protect
1019 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1020 \noexpand\newcommand
1021 \expandafter\noexpand\csname text##1\endcsname{%
1022 \noexpand\foreignlanguage{##2}}}
1023 \bb1@tempc}%
1024 \bb1@for\bb1@tempa\bb1@tempa{%
1025 \expandafter\bb1@tempb\bb1@tempa\@{}}

```

3.10. Compatibility with language.def

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

```
1026 \bbl@trace{Compatibility with language.def}
1027 \ifx\directlua@undefined\else
1028   \ifx\bbl@luapatterns@undefined
1029     \input luababel.def
1030   \fi
1031 \fi
1032 \ifx\bbl@languages@undefined
1033   \ifx\directlua@undefined
1034     \openin0 = language.def
1035     \ifeof0
1036       \closein0
1037       \message{I couldn't find the file language.def}
1038     \else
1039       \closein0
1040       \begingroup
1041         \def\addlanguage#1#2#3#4#5{%
1042           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1043             \global\expandafter\let\csname l@#1\expandafter\endcsname
1044               \csname lang@#1\endcsname
1045           \fi}%
1046         \def\uselanguage#1{%
1047           \input language.def
1048         \endgroup
1049       \fi
1050     \fi
1051   \chardef\l@english\z@
1052 \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.

```
1053 \def\addto#1#2{%
1054   \ifx#1@undefined
1055     \def#1{#2}%
1056   \else
1057     \ifx#1\relax
1058       \def#1{#2}%
1059     \else
1060       {\toks@\expandafter{#1#2}%
1061        \xdef#1{\the\toks@}}%
1062     \fi
1063   \fi}
```

3.11. 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.

```
1064 \bbl@trace{Hooks}
1065 \newcommand\AddBabelHook[3][[]]{%
1066   \bbl@ifunset\bbl@hk@#2{\EnableBabelHook{#2}}{}%
1067   \def\bbl@tempa##1,##2,##3\@empty{\def\bbl@tempb{##2}}%
1068   \expandafter\bbl@tempa\bbl@evargs,##3,\@empty
1069   \bbl@ifunset\bbl@ev@#2@#3@#1{%
1070     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1071     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1072   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
```

```

1073 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1074 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1075 \def\bbl@usehooks{\bbl@usehooks@lang\languagename}
1076 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1077   \ifx\UseHook\undefined\else\UseHook{babel/*/#2}\fi
1078   \def\bbl@elth##1{%
1079     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1080     \bbl@cs{ev@#2@}%
1081     \ifx\languagename\undefined\else % Test required for Plain (?)
1082       \ifx\UseHook\undefined\else\UseHook{babel/#1/#2}\fi
1083       \def\bbl@elth##1{%
1084         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1#3}}%
1085         \bbl@cs{ev@#2@#1}%
1086       \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).

```

1087 \def\bbl@evargs{% <- don't delete this comma
1088   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1089   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1090   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1091   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1092   beforestart=0,languagename=2,begindocument=1}
1093 \ifx\NewHook\undefined\else % Test for Plain (?)
1094   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}\NewHook{babel/*/#1}}
1095   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1096 \fi

```

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

```

1097 \providecommand\PassOptionsToLocale[2]{%
1098   \bbl@csarg\bbl@add@list{passto@#2}{#1}}

```

3.12. 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`.

```

1099 \bbl@trace{Macros for setting language files up}
1100 \def\bbl@ldfinit{%
1101   \let\bbl@screset\@empty
1102   \let\BabelStrings\bbl@opt@string
1103   \let\BabelOptions\@empty
1104   \let\BabelLanguages\relax
1105   \ifx\originalTeX\undefined
1106     \let\originalTeX\@empty
1107   \else
1108     \originalTeX

```

```

1109 \fi}
1110 \def\LdfInit#1#2{%
1111 \chardef\atcatcode=\catcode`\@
1112 \catcode`\@=11\relax
1113 \chardef\eqcatcode=\catcode`\=
1114 \catcode`\=12\relax
1115 \ifpackagewith{babel}{ensureinfo=off}{}%
1116 {\ifx\InputIfFileExists\@undefined\else
1117 \bbl@ifunset{bbl@lname@#1}%
1118 {\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1119 \def\language#1{%
1120 \bbl@id@assign
1121 \bbl@load@info{#1}}}%
1122 }%
1123 \fi}%
1124 \expandafter\if\expandafter\@backslashchar
1125 \expandafter\@car\string#2\@nil
1126 \ifx#2\@undefined\else
1127 \ldf@quit{#1}%
1128 \fi
1129 \else
1130 \expandafter\ifx\csname#2\endcsname\relax\else
1131 \ldf@quit{#1}%
1132 \fi
1133 \fi
1134 \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.

```

1135 \def\ldf@quit#1{%
1136 \expandafter\main@language\expandafter{#1}%
1137 \catcode`\@=\atcatcode \let\atcatcode\relax
1138 \catcode`\=\eqcatcode \let\eqcatcode\relax
1139 \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.

```

1140 \def\bbl@afterldf{%
1141 \bbl@afterlang
1142 \let\bbl@afterlang\relax
1143 \let\BabelModifiers\relax
1144 \let\bbl@screset\relax}%
1145 \def\ldf@finish#1{%
1146 \loadlocalcfg{#1}%
1147 \bbl@afterldf
1148 \expandafter\main@language\expandafter{#1}%
1149 \catcode`\@=\atcatcode \let\atcatcode\relax
1150 \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 *TeX*.

```

1151 \@onlypreamble\LdfInit
1152 \@onlypreamble\ldf@quit
1153 \@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.

```

1154 \def\main@language#1{%
1155   \def\bbl@main@language{#1}%
1156   \let\language\main@language
1157   \let\localename\bbl@main@language
1158   \let\mainlocalename\bbl@main@language
1159   \bbl@id@assign
1160   \ifcase\bbl@engine\or
1161     \ifx\setattribute\undefined\else
1162       \setattribute\bbl@attr@locale\localeid
1163     \fi
1164   \fi
1165   \bbl@patterns{\language}%

```

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

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

```

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

```

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

```

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

```

3.13. 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.

```

1192 \bbl@trace{Shorhands}
1193 \def\bbl@withactive#1#2{%
1194   \begingroup
1195     \lccode`~=`#2\relax
1196     \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 L^AT_EX 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.

```

1197 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1198 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1199 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1200 \ifx\nfss@catcodes\@undefined\else
1201 \begingroup
1202 \catcode`#1\active
1203 \nfss@catcodes
1204 \ifnum\catcode`#1=\active
1205 \endgroup
1206 \bbl@add\nfss@catcodes{\@makeother#1}%
1207 \else
1208 \endgroup
1209 \fi
1210 \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).

```

1211 \def\bbl@active@def#1#2#3#4{%
1212 \namedef{#3#1}{%
1213 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1214 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1215 \else
1216 \bbl@afterfi\csname#2@sh@#1\endcsname
1217 \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.

```

1218 \long\namedef{#3@arg#1}##1{%
1219 \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1220 \bbl@afterelse\csname#4#1\endcsname##1%
1221 \else
1222 \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1223 \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.

```

1224 \def\initiate@active@char#1{%
1225 \bbl@ifunset{active@char\string#1}%
1226 {\bbl@withactive
1227 {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1228 {}}

```

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).

```

1229 \def\@initiate@active@char#1#2#3{%
1230   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1231   \ifx#1\@undefined
1232     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1233   \else
1234     \bbl@csarg\let{oridef@#2}#1%
1235     \bbl@csarg\edef{oridef@#2}{%
1236       \let\noexpand#1%
1237       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1238   \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*").

```

1239   \ifx#1#3\relax
1240     \expandafter\let\csname normal@char#2\endcsname#3%
1241   \else
1242     \bbl@info{Making #2 an active character}%
1243     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1244       \@namedef{normal@char#2}{%
1245         \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1246       \else
1247         \@namedef{normal@char#2}{#3}%
1248       \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).

```

1249   \bbl@restoreactive{#2}%
1250   \AtBeginDocument{%
1251     \catcode`#2\active
1252     \if@filesw
1253       \immediate\write\@mainaux{\catcode`\string#2\active}%
1254     \fi}%
1255   \expandafter\bbl@add@special\csname#2\endcsname
1256   \catcode`#2\active
1257 \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⟩).

```

1258   \let\bbl@tempa\@firstoftwo
1259   \if\string^#2%
1260     \def\bbl@tempa{\noexpand\textormath}%
1261   \else
1262     \ifx\bbl@mathnormal\@undefined\else
1263       \let\bbl@tempa\bbl@mathnormal
1264     \fi
1265   \fi
1266   \expandafter\edef\csname active@char#2\endcsname{%
1267     \bbl@tempa
1268     {\noexpand\if@safe@actives
1269       \noexpand\expandafter

```

```

1270      \expandafter\noexpand\csname normal@char#2\endcsname
1271      \noexpand\else
1272      \noexpand\expandafter
1273      \expandafter\noexpand\csname bbl@doactive#2\endcsname
1274      \noexpand\fi}%
1275      {\expandafter\noexpand\csname normal@char#2\endcsname}}}%
1276      \bbl@csarg\edef{doactive#2}{%
1277      \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

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

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

```

1278      \bbl@csarg\edef{active@#2}{%
1279      \noexpand\active@prefix\noexpand#1%
1280      \expandafter\noexpand\csname active@char#2\endcsname}%
1281      \bbl@csarg\edef{normal@#2}{%
1282      \noexpand\active@prefix\noexpand#1%
1283      \expandafter\noexpand\csname normal@char#2\endcsname}%
1284      \bbl@ncarg\let#1{\bbl@normal@#2}%

```

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

```

1285      \bbl@active@def#2\user@group{user@active}{language@active}%
1286      \bbl@active@def#2\language@group{language@active}{system@active}%
1287      \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1288      \expandafter\edef\csname\user@group @sh#2@@\endcsname
1289      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1290      \expandafter\edef\csname\user@group @sh#2@\string\protect'\endcsname
1291      {\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.

```

1292      \if\string'#2%
1293      \let\prim@s\bbl@prim@s
1294      \let\active@math@prime#1%
1295      \fi
1296      \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

1297      (< *More package options>) ▶
1298      \DeclareOption{math=active}{}
1299      \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1300      (< /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*.

```

1301      \ifpackagewith{babel}{KeepShorthandsActive}%
1302      {\let\bbl@restoreactive@gobble}%
1303      {\def\bbl@restoreactive#1{%
1304      \bbl@exp{%
1305      \\AfterBabelLanguage\\CurrentOption

```

```

1306      {\catcode`#1=\the\catcode`#1\relax}%
1307      \\AtEndOfPackage
1308      {\catcode`#1=\the\catcode`#1\relax}}}%
1309      \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.

```

1310 \def\bbl@sh@select#1#2{%
1311   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1312     \bbl@afterelse\bbl@scndcs
1313   \else
1314     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1315   \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.

```

1316 \begingroup
1317 \bbl@ifunset{ifincsname}
1318 {\gdef\active@prefix#1{%
1319   \ifx\protect\@typeset@protect
1320   \else
1321     \ifx\protect\@unexpandable@protect
1322       \noexpand#1%
1323     \else
1324       \protect#1%
1325     \fi
1326     \expandafter\@gobble
1327   \fi}}
1328 {\gdef\active@prefix#1{%
1329   \ifincsname
1330     \string#1%
1331     \expandafter\@gobble
1332   \else
1333     \ifx\protect\@typeset@protect
1334     \else
1335       \ifx\protect\@unexpandable@protect
1336         \noexpand#1%
1337       \else
1338         \protect#1%
1339       \fi
1340       \expandafter\expandafter\expandafter\@gobble
1341     \fi
1342   \fi}}
1343 \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`).

```

1344 \newif\if@safe@actives
1345 \@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.

```
1346 \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`.

```
1347 \chardef\bbl@activated\z@
1348 \def\bbl@activate#1{%
1349   \chardef\bbl@activated\@ne
1350   \bbl@withactive{\expandafter\let\expandafter}#1%
1351   \csname bbl@active@\string#1\endcsname}
1352 \def\bbl@deactivate#1{%
1353   \chardef\bbl@activated\tw@
1354   \bbl@withactive{\expandafter\let\expandafter}#1%
1355   \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs

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

```
1356 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1357 \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 $\TeX$ code in text mode, (2) the string for `hyperref`, (3) the $\TeX$ code in math mode, and (4), which is currently ignored, but it’s meant for a string in math mode, like a minus sign instead of an hyphen (currently `hyperref` doesn’t discriminate the mode). This macro may be used in `ldf` files.

```
1358 \def\babel@texpdf#1#2#3#4{%
1359   \ifx\texorpdfstring\undefined
1360     \textormath{#1}{#3}%
1361   \else
1362     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1363     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1364   \fi}
1365 %
1366 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1367 \def\@decl@short#1#2#3\@nil#4{%
1368   \def\bbl@tempa{#3}%
1369   \ifx\bbl@tempa\@empty
1370     \expandafter\let\csname #1@sh@\string#2\endcsname\bbl@scndcs
1371     \bbl@ifunset{#1@sh@\string#2@}{}%
1372     {\def\bbl@tempa{#4}%
1373      \expandafter\ifx\csname#1@sh@\string#2\endcsname\bbl@tempa
1374      \else
1375        \bbl@info
1376        {Redefining #1 shorthand \string#2\\%
1377         in language \CurrentOption}%
1378      \fi}%
1379   \@namedef{#1@sh@\string#2@}{#4}%
1380 \else
1381   \expandafter\let\csname #1@sh@\string#2\endcsname\bbl@firstcs
1382   \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1383   {\def\bbl@tempa{#4}%
1384    \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
```

```

1385     \else
1386     \bbl@info
1387     {Redefining #1 shorthand \string#2\string#3\\%
1388     in language \CurrentOption}%
1389     \fi}%
1390     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1391 \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.

```

1392 \def\textormath{%
1393 \ifmmode
1394 \expandafter\@secondoftwo
1395 \else
1396 \expandafter\@firstoftwo
1397 \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’.

```

1398 \def\user@group{user}
1399 \def\language@group{english}
1400 \def\system@group{system}

```

\usesshorthands 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.

```

1401 \def\usesshorthands{%
1402 \ifstar\bbl@useseshs{\bbl@useseshx{}}
1403 \def\bbl@useseshs#1{%
1404 \bbl@useseshx
1405 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1406 {#1}}
1407 \def\bbl@useseshx#1#2{%
1408 \bbl@ifshorthand{#2}%
1409 {\def\user@group{user}%
1410 \initiate@active@char{#2}%
1411 #1%
1412 \bbl@activate{#2}}%
1413 {\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.

```

1414 \def\user@language@group{user@\language@group}
1415 \def\bbl@set@user@generic#1#2{%
1416 \bbl@ifunset{user@generic@active#1}%
1417 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1418 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1419 \expandafter\edef\csname#2@sh@#1@\endcsname{%
1420 \expandafter\noexpand\csname normal@char#1\endcsname}%
1421 \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1422 \expandafter\noexpand\csname user@active#1\endcsname}}%
1423 \@empty}
1424 \newcommand\defineshorthand[3][user]{%
1425 \edef\bbl@tempa{\zap@space#1 \@empty}%

```

```

1426 \bbl@for\bbl@tempb\bbl@tempa{%
1427 \if*\expandafter\@car\bbl@tempb\@nil
1428 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1429 \@expandtwoargs
1430 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1431 \fi
1432 \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.

```

1433 \def\languageshorthands#1{%
1434 \bbl@ifsamestring{none}{#1}}{%
1435 \bbl@once{short-\localename-#1}{%
1436 \bbl@info{'\localename' activates '#1' shorthands.\@Reported}}}%
1437 \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".

```

1438 \def\aliasshorthand#1#2{%
1439 \bbl@ifshorthand{#2}%
1440 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1441 \ifx\document\@notprerr
1442 \@notshorthand{#2}%
1443 \else
1444 \initiate@active@char{#2}%
1445 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1446 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1447 \bbl@activate{#2}%
1448 \fi
1449 \fi}%
1450 {\bbl@error{shorthand-is-off}{#2}{}}}

```

\@notshorthand

```

1451 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}

```

\shorthandon

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

```

1452 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1453 \DeclareRobustCommand*\shorthandoff{%
1454 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1455 \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.

```

1456 \def\bbl@switch@sh#1#2{%
1457 \ifx#2\@nnil\else
1458 \bbl@ifunset{\bbl@active@\string#2}%
1459 {\bbl@error{not-a-shorthand-b}{#2}{}}%
1460 {\ifcase#1% off, on, off*
1461 \catcode`#212\relax
1462 \or

```

```

1463 \catcode`#2\active
1464 \bbl@ifunset{bbl@shdef@\string#2}%
1465 {}%
1466 {\bbl@withactive{\expandafter\let\expandafter}#2%
1467 \csname bbl@shdef@\string#2\endcsname
1468 \bbl@csarg\let{shdef@\string#2}\relax}%
1469 \ifcase\bbl@activated\or
1470 \bbl@activate{#2}%
1471 \else
1472 \bbl@deactivate{#2}%
1473 \fi
1474 \or
1475 \bbl@ifunset{bbl@shdef@\string#2}%
1476 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1477 {}%
1478 \csname bbl@oricat@\string#2\endcsname
1479 \csname bbl@oridef@\string#2\endcsname
1480 \fi}%
1481 \bbl@afterfi\bbl@switch@sh#1%
1482 \fi}

```

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

```

1483 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1484 \def\bbl@putsh#1{%
1485 \bbl@ifunset{bbl@active@\string#1}%
1486 {\bbl@putsh@i#1\@empty\@nnil}%
1487 {\csname bbl@active@\string#1\endcsname}}
1488 \def\bbl@putsh@i#1#2\@nnil{%
1489 \csname\language@group @sh@\string#1@%
1490 \ifx\@empty#2\else\string#2@\fi\endcsname}
1491 %
1492 \ifx\bbl@opt@shorthands\@nnil\else
1493 \let\bbl@s@initiate@active@char\initiate@active@char
1494 \def\initiate@active@char#1{%
1495 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1496 \let\bbl@s@switch@sh\bbl@switch@sh
1497 \def\bbl@switch@sh#1#2{%
1498 \ifx#2\@nnil\else
1499 \bbl@afterfi
1500 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1501 \fi}
1502 \let\bbl@s@activate\bbl@activate
1503 \def\bbl@activate#1{%
1504 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1505 \let\bbl@s@deactivate\bbl@deactivate
1506 \def\bbl@deactivate#1{%
1507 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1508 \fi

```

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

```

1509 \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.

```

1510 \def\bbl@prim@s{%
1511 \prime\futurelet\@let@token\bbl@pr@m@s}
1512 \def\bbl@if@primes#1#2{%
1513 \ifx#1\@let@token

```

```

1514 \expandafter\@firstoftwo
1515 \else\ifx#2\@let@token
1516 \bbl@afterelse\expandafter\@firstoftwo
1517 \else
1518 \bbl@afterfi\expandafter\@secondoftwo
1519 \fi\fi}
1520 \begingroup
1521 \catcode\^=7 \catcode\*= \active \lccode\*= \^
1522 \catcode\'=12 \catcode\"= \active \lccode\"= \'
1523 \lowercase{%
1524 \gdef\bbl@pr@ms{%
1525 \bbl@if@primes''%
1526 \pr@@s
1527 {\bbl@if@primes*\pr@@t\egroup}}
1528 \endgroup

```

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

```

1529 \initiate@active@char{~}
1530 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1531 \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.

```

1532 \expandafter\def\csname OT1dqpos\endcsname{127}
1533 \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

```

1534 \ifx\f@encoding\undefined
1535 \def\f@encoding{OT1}
1536 \fi

```

3.14. 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.

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.

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.

```

1537 \bbl@trace{Language attributes}
1538 \newcommand\languageattribute[2]{%
1539 \def\bbl@tempc{#1}%
1540 \bbl@fixname\bbl@tempc
1541 \bbl@iflanguage\bbl@tempc{%
1542 \bbl@vforeach{#2}{%
1543 \ifx\bbl@known@attrs\undefined
1544 \in@false
1545 \else
1546 \bbl@xin@{\,\bbl@tempc-##1,}{\,\bbl@known@attrs,}%
1547 \fi

```

```

1548 \ifin@
1549 \bbl@warning{%
1550 You have more than once selected the attribute '##1'\%
1551 for language #1. Reported}%
1552 \else
1553 \bbl@info{Activated '##1' attribute for\%
1554 '\bbl@tempc'. Reported}%
1555 \bbl@exp{%
1556 \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1557 \edef\bbl@tempa{\bbl@tempc-##1}%
1558 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes
1559 {\csname\bbl@tempc @attr##1\endcsname}%
1560 {\bbl@error{unknown-attribute}{\bbl@tempc}{##1}}}%
1561 \fi}}
1562 \@onlypreamble\languageattribute

```

\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}`.

```

1563 \def\bbl@declare@ttribute#1#2#3{%
1564 \bbl@xin@{,#2,}{,\BabelModifiers,}%
1565 \ifin@
1566 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1567 \fi
1568 \bbl@add@list\bbl@attributes{#1-#2}%
1569 \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.

```

1570 \def\bbl@ifattributeset#1#2#3#4{%
1571 \ifx\bbl@known@attribs\@undefined
1572 \in@false
1573 \else
1574 \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1575 \fi
1576 \ifin@
1577 \bbl@afterelse#3%
1578 \else
1579 \bbl@afterfi#4%
1580 \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.

```

1581 \def\bbl@ifknown@ttrib#1#2{%
1582 \let\bbl@tempa\@secondoftwo
1583 \bbl@loopx\bbl@tempb{#2}{%
1584 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1585 \ifin@
1586 \let\bbl@tempa\@firstoftwo
1587 \else
1588 \fi}%
1589 \bbl@tempa}

```

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

```

1590 \def\bbl@clear@ttribs{%
1591   \ifx\bbl@attributes\undefined\else
1592     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1593       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1594     \let\bbl@attributes\undefined
1595   \fi}
1596 \def\bbl@clear@ttrib#1-#2.{%
1597   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1598 \AtBeginDocument{\bbl@clear@ttribs}

```

3.15. 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.

```

1599 \bbl@trace{Macros for saving definitions}
1600 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1601 \newcount\babel@savecnt
1602 \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.

```

1603 \def\babel@save#1{%
1604   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1605   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1606     \expandafter{\expandafter,\bbl@savextrs,}}%
1607   \expandafter\in@\bbl@tempa
1608   \ifin@else
1609     \bbl@add\bbl@savextrs{,{#1,}}%
1610     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1611     \toks@{\expandafter{\originalTeX\let#1=}}%
1612     \bbl@exp{%
1613       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1614     \advance\babel@savecnt\@ne
1615   \fi}
1616 \def\babel@savevariable#1{%
1617   \toks@{\expandafter{\originalTeX #1=}}%
1618   \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`.

```

1619 \def\bb@l@redefine#1{%
1620   \edef\bb@tempa{\bb@stripslash#1}%
1621   \expandafter\let\csname org@\bb@tempa\endcsname#1%
1622   \expandafter\def\csname\bb@tempa\endcsname}
1623 \@onlypreamble\bb@l@redefine

```

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

```

1624 \def\bb@l@redefine@long#1{%
1625   \edef\bb@tempa{\bb@stripslash#1}%
1626   \expandafter\let\csname org@\bb@tempa\endcsname#1%
1627   \long\expandafter\def\csname\bb@tempa\endcsname}
1628 \@onlypreamble\bb@l@redefine@long

```

\bb@l@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.

```

1629 \def\bb@l@redefineroobust#1{%
1630   \edef\bb@tempa{\bb@stripslash#1}%
1631   \bb@ifunset{\bb@tempa\space}%
1632     {\expandafter\let\csname org@\bb@tempa\endcsname#1%
1633       \bb@exp{\def\#1{\protect\<\bb@tempa\space>}}}%
1634     {\bb@exp{\let\<org@\bb@tempa>\<\bb@tempa\space>}}}%
1635     \@namedef{\bb@tempa\space}}
1636 \@onlypreamble\bb@l@redefineroobust

```

3.16. French spacing

\bb@l@frenchspacing

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

```

1637 \def\bb@l@frenchspacing{%
1638   \ifnum\the\sfcodes\<.\<=\@m
1639     \let\bb@l@nonfrenchspacing\relax
1640   \else
1641     \frenchspacing
1642     \let\bb@l@nonfrenchspacing\nonfrenchspacing
1643   \fi}
1644 \let\bb@l@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.

```

1645 \let\bb@l@elt\relax
1646 \edef\bb@l@fs@chars{%
1647   \bb@elt{\string.}\@m{3000}\bb@elt{\string?}\@m{3000}%
1648   \bb@elt{\string!}\@m{3000}\bb@elt{\string:}\@m{2000}%
1649   \bb@elt{\string;}\@m{1500}\bb@elt{\string,}\@m{1250}}
1650 \def\bb@l@pre@fs{%
1651   \def\bb@l@elt##1##2##3{\sfcode`##1=\the\sfcodes`##1\relax}%
1652   \edef\bb@l@save@sfcodes{\bb@l@fs@chars}%
1653 \def\bb@l@post@fs{%
1654   \bb@l@save@sfcodes
1655   \edef\bb@l@tempa{\bb@cl{frspc}}%
1656   \edef\bb@l@tempa{\expandafter\@car\bb@l@tempa\@nil}%
1657   \if u\bb@l@tempa % do nothing
1658   \else\if n\bb@l@tempa % non french
1659     \def\bb@l@elt##1##2##3{%
1660       \ifnum\sfcodes`##1=##2\relax
1661       \babel@savevariable{\sfcode`##1}%

```

3.17. Hyphens

```

1673 \bbl@trace{Hyphens}
1674 \@onlypreamble\babelhyphenation
1675 \AtEndOfPackage{%
1676   \newcommand\babelhyphenation[2][\@empty]{%
1677     \ifx\bbl@hyphenation@\relax
1678       \let\bbl@hyphenation@\@empty
1679     \fi
1680     \ifx\bbl@hyphlist\@empty\else
1681       \bbl@warning{%
1682         You must not intermingle \string\selectlanguage\space and\\%
1683         \string\babelhyphenation\space or some exceptions will not\\%
1684         be taken into account. Reported}%
1685     \fi
1686     \ifx\@empty#1%
1687       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1688     \else
1689       \bbl@vforeach{#1}{%
1690         \def\bbl@tempa{##1}%
1691         \bbl@fixname\bbl@tempa
1692         \bbl@iflanguage\bbl@tempa{%
1693           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1694             \bbl@ifunset{\bbl@hyphenation@\bbl@tempa}%
1695             {}%
1696             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1697             #2}}}%
1698       \fi}}

```

```

1699 \ifx\NewDocumentCommand\undefined\else
1700   \NewDocumentCommand\babelhyphenmins{sommo}{%
1701     \IfNoValueTF{#2}%
1702       {\protected@edef\bbl@hyphenmins@{\set@hyphenmins{#3}{#4}}}%
1703       \IfValueT{#5}{%
1704         \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1705       \IfBooleanT{#1}{%
1706         \lefthyphenmin=#3\relax
1707         \righthyphenmin=#4\relax
1708         \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1709   {\edef\bbl@tempb{\zap@space#2 \@empty}%
1710     \bbl@for\bbl@tempa\bbl@tempb{%
1711       \@namedef{\bbl@hyphenmins@\bbl@tempa}{\set@hyphenmins{#3}{#4}}%
1712       \IfValueT{#5}{%
1713         \@namedef{\bbl@hyphenatmin@\bbl@tempa}{\hyphenationmin=#5\relax}}}%
1714     \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}}{}{}{}{}{}%

```

1715 \fi

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

```
1716 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1717 \def\bbl@t@one{T1}
1718 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
```

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

```
1719 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1720 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1721 \def\bbl@hyphen{%
1722   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1723 \def\bbl@hyphen@i#1#2{%
1724   \lowercase{\bbl@ifunset{\bbl@hy@#1#2\@empty}}%
1725   {\csname bbl@#1usehyphen\endcsname\discretionary{#2}{#2}}}%
1726   {\lowercase{\csname bbl@hy@#1#2\@empty\endcsname}}}
```

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

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

```
1727 \def\bbl@usehyphen#1{%
1728   \leavevmode
1729   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1730   \nobreak\hskip\z@skip}
1731 \def\bbl@@usehyphen#1{%
1732   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1733 \def\bbl@hyphenchar{%
1734   \ifnum\hyphenchar\font=\m@ne
1735     \babelnullhyphen
1736   \else
1737     \char\hyphenchar\font
1738   \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.

```
1739 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1740 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1741 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1742 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1743 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1744 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1745 \def\bbl@hy@repeat{%
1746   \bbl@usehyphen%
1747   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1748 \def\bbl@hy@@repeat{%
1749   \bbl@usehyphen%
1750   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1751 \def\bbl@hy@empty{\hskip\z@skip}
1752 \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.

```
1753 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}
```

3.18. 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.

```
1754 \bbl@trace{Multiencoding strings}
1755 \def\bbl@tglobal#1{\global\let#1#1}
```

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

```
1756 ((*More package options)) ▶
1757 \DeclareOption{nocase}{%
1758 (/More package options)
```

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

```
1759 ((*More package options)) ▶
1760 \let\bbl@opt@strings\@nnil % accept strings=value
1761 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1762 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1763 \def\BabelStringsDefault{generic}
1764 (/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.

```
1765 \@onlypreamble\StartBabelCommands
1766 \def\StartBabelCommands{%
1767   \begingroup
1768   \@tempcnta="7F
1769   \def\bbl@tempa{%
1770     \ifnum\@tempcnta>"FF\else
1771       \catcode\@tempcnta=11
1772       \advance\@tempcnta\@ne
1773       \expandafter\bbl@tempa
1774     \fi}%
1775   \bbl@tempa
1776   <@Macros local to BabelCommands@>
1777   \def\bbl@provstring##1##2{%
1778     \providecommand##1{##2}%
1779     \bbl@tglobal##1}%
1780   \global\let\bbl@scafter\@empty
1781   \let\StartBabelCommands\bbl@startcmds
1782   \ifx\BabelLanguages\relax
1783     \let\BabelLanguages\CurrentOption
1784   \fi
1785   \begingroup
1786   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1787   \StartBabelCommands}
1788 \def\bbl@startcmds{%
1789   \ifx\bbl@screset\@nnil\else
1790     \bbl@usehooks{stopcommands}{}%
1791   \fi
1792   \endgroup
1793   \begingroup
1794   \@ifstar
1795     {\ifx\bbl@opt@strings\@nnil
1796       \let\bbl@opt@strings\BabelStringsDefault
1797     \fi
1798     \bbl@startcmds@i}%
1799   \bbl@startcmds@i}
1800 \def\bbl@startcmds@i#1#2{%
1801   \edef\bbl@L{\zap@space#1 \@empty}%
```

```

1802 \edef\bbl@G{\zap@space#2 \@empty}%
1803 \bbl@startcmds@ii}
1804 \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.

```

1805 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1806 \let\SetString\@gobbletwo
1807 \let\bbl@stringdef\@gobbletwo
1808 \let\AfterBabelCommands\@gobble
1809 \ifx\@empty#1%
1810 \def\bbl@sc@label{generic}%
1811 \def\bbl@encstring##1##2{%
1812 \ProvideTextCommandDefault##1{##2}%
1813 \bbl@tglobal##1%
1814 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1815 \let\bbl@sctest\in@true
1816 \else
1817 \let\bbl@sc@charset\space % <- zapped below
1818 \let\bbl@sc@fontenc\space % <- " "
1819 \def\bbl@tempa##1=##2\@nil{%
1820 \bbl@csarg\edef\sc{\zap@space##1 \@empty}{##2 }}%
1821 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1822 \def\bbl@tempa##1 ##2{% space -> comma
1823 ##1%
1824 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1825 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1826 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1827 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1828 \def\bbl@encstring##1##2{%
1829 \bbl@foreach\bbl@sc@fontenc{%
1830 \bbl@ifunset{T@###1}%
1831 }%
1832 {\ProvideTextCommand##1{####1}{##2}%
1833 \bbl@tglobal##1%
1834 \expandafter
1835 \bbl@tglobal\csname###1\string##1\endcsname}}}%
1836 \def\bbl@sctest{%
1837 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
1838 \fi
1839 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1840 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1841 \let\AfterBabelCommands\bbl@aftercmds
1842 \let\SetString\bbl@setstring
1843 \let\bbl@stringdef\bbl@encstring
1844 \else % i.e., strings=value
1845 \bbl@sctest
1846 \ifin@
1847 \let\AfterBabelCommands\bbl@aftercmds
1848 \let\SetString\bbl@setstring
1849 \let\bbl@stringdef\bbl@provstring
1850 \fi\fi\fi
1851 \bbl@scswitch
1852 \ifx\bbl@G\@empty
1853 \def\SetString##1##2{%
1854 \bbl@error{missing-group}{##1}{}}}%

```

```

1855 \fi
1856 \ifx\@empty#1%
1857 \bbl@usehooks{defaultcommands}{}%
1858 \else
1859 \@expandtwoargs
1860 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1861 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\group\language` 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\language` 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).

```

1862 \def\bbl@forlang#1#2{%
1863 \bbl@for#1\bbl@L{%
1864 \bbl@xin@{,#1,}{,\BabelLanguages,}%
1865 \ifin#2\relax\fi}}
1866 \def\bbl@scswitch{%
1867 \bbl@forlang\bbl@tempa{%
1868 \ifx\bbl@G\@empty\else
1869 \ifx\SetString@gobbletwo\else
1870 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1871 \bbl@xin@{\bbl@GL,}{,\bbl@screset,}%
1872 \ifin\else
1873 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1874 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1875 \fi
1876 \fi
1877 \fi}}
1878 \AtEndOfPackage{%
1879 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1880 \let\bbl@scswitch\relax}
1881 \@onlypreamble\EndBabelCommands
1882 \def\EndBabelCommands{%
1883 \bbl@usehooks{stopcommands}{}%
1884 \endgroup
1885 \endgroup
1886 \bbl@scafter}
1887 \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.

```

1888 \def\bbl@setstring#1#2{e.g., \prefacename{<string>}
1889 \bbl@forlang\bbl@tempa{%
1890 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1891 \bbl@ifunset{\bbl@LC}{e.g., \germanchaptername
1892 {\bbl@exp}%
1893 \global\let\bbl@add\<\bbl@G\bbl@tempa{\bbl@scset\#1\<\bbl@LC>}}}%
1894 {}%
1895 \def\BabelString{#2}%
1896 \bbl@usehooks{stringprocess}{}%
1897 \expandafter\bbl@stringdef
1898 \csname\bbl@LC\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`.

```
1899 \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.

```
1900 (<(*Macros local to BabelCommands)) ▶
1901 \def\SetStringLoop##1##2{%
1902   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1903   \count@\z@
1904   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1905     \advance\count@\@ne
1906     \toks@\expandafter{\bbl@tempa}%
1907     \bbl@exp{%
1908       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1909       \count@=\the\count@\relax}}}%
1910 (</Macros local to BabelCommands))
```

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

```
1911 \def\bbl@aftercmds#1{%
1912   \toks@\expandafter{\bbl@scafter#1}%
1913   \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.

```
1914 (<(*Macros local to BabelCommands)) ▶
1915   \newcommand\SetCase[3][]{%
1916     \def\bbl@tempa####1####2{%
1917       \ifx####1@empty\else
1918         \bbl@carg\bbl@add{extras\CurrentOption}{%
1919           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1920           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1921           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1922           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}%
1923           \expandafter\bbl@tempa
1924         \fi}%
1925     \bbl@tempa##1@empty@empty
1926     \bbl@carg\bbl@toglobal{extras\CurrentOption}}%
1927 (</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.

```
1928 (<(*Macros local to BabelCommands)) ▶
1929   \newcommand\SetHyphenMap[1]{%
1930     \bbl@forlang\bbl@tempa{%
1931       \expandafter\bbl@stringdef
1932       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1933 (</Macros local to BabelCommands))
```

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

```
1934 \newcommand\BabelLower[2]{% one to one.
1935   \ifnum\lccode#1=#2\else
1936     \babel@savevariable{\lccode#1}%
1937     \lccode#1=#2\relax
1938   \fi}
1939 \newcommand\BabelLowerMM[4]{% many-to-many
1940   \@tempcnta=#1\relax
1941   \@tempcntb=#4\relax
1942   \def\bbl@tempa{%
1943     \ifnum\@tempcnta>#2\else
1944       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1945       \advance\@tempcnta#3\relax
```

```

1946     \advance\@tempcntb#3\relax
1947     \expandafter\bbbl@tempa
1948     \fi}%
1949     \bbbl@tempa}
1950 \newcommand\BabelLowerM0[4]{% many-to-one
1951   \@tempcnta=#1\relax
1952   \def\bbbl@tempa{%
1953     \ifnum\@tempcnta>#2\else
1954       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1955       \advance\@tempcnta#3
1956       \expandafter\bbbl@tempa
1957     \fi}%
1958   \bbbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1959 ((*More package options)) ▶
1960 \DeclareOption{hyphenmap=off}{\chardef\bbbl@opt@hyphenmap\z@}
1961 \DeclareOption{hyphenmap=first}{\chardef\bbbl@opt@hyphenmap\@ne}
1962 \DeclareOption{hyphenmap=select}{\chardef\bbbl@opt@hyphenmap\tw@}
1963 \DeclareOption{hyphenmap=other}{\chardef\bbbl@opt@hyphenmap\thr@@}
1964 \DeclareOption{hyphenmap=other*}{\chardef\bbbl@opt@hyphenmap4\relax}
1965 (/*More package options))

```

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

```

1966 \AtEndOfPackage{%
1967   \ifx\bbbl@opt@hyphenmap\undefined
1968     \bbbl@xin@{,}{\bbbl@language@opts}%
1969     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1970   \fi}

```

3.19. 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.

```

1971 \newcommand\setlocalecaption{%
1972   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1973 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1974   \bbbl@trim@def\bbbl@tempa{#2}%
1975   \bbbl@xin@{.template}{\bbbl@tempa}%
1976   \ifin@
1977     \bbbl@ini@captions@template{#3}{#1}%
1978   \else
1979     \edef\bbbl@tempd{%
1980       \expandafter\expandafter\expandafter
1981       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1982     \bbbl@xin@
1983       {\expandafter\string\csname #2name\endcsname}%
1984       {\bbbl@tempd}%
1985     \ifin@ % Renew caption
1986       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
1987       \ifin@
1988         \bbbl@exp{%
1989           \\bbbl@ifsamestring{\bbbl@tempa}{\language name}%
1990           {\bbbl@scset\<#2name>\<#1#2name>}}%
1991         {}}%
1992     \else % Old way converts to new way
1993       \bbbl@ifunset{#1#2name}%
1994       {\bbbl@exp{%
1995         \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1996         \\bbbl@ifsamestring{\bbbl@tempa}{\language name}%
1997         {\def\<#2name>{\<#1#2name>}}%
1998         {}}}}
1999     {}%

```

```

2000     \fi
2001   \else
2002     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2003     \ifin@ % New way
2004       \bbl@exp{%
2005         \\bbl@add<captions#1>{\bbl@scset<#2name>\<#1#2name>}%
2006         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2007         {\bbl@scset<#2name>\<#1#2name>}%
2008         }%
2009       \else % Old way, but defined in the new way
2010         \bbl@exp{%
2011           \\bbl@add<captions#1>{\def<#2name>{\<#1#2name>}}%
2012           \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2013           {\def<#2name>{\<#1#2name>}}%
2014           }%
2015       \fi%
2016   \fi
2017   \@namedef{#1#2name}{#3}%
2018   \toks@{\expandafter\bbl@captionslist}%
2019   \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2020   \ifin@else
2021     \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2022     \bbl@tglobal\bbl@captionslist
2023   \fi
2024 \fi}

```

3.20. 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.

```

2025 \bbl@trace{Macros related to glyphs}
2026 \def\set@low@box#1{\setbox\tw@{#1}\setbox\z@{\hbox{#1}}%
2027   \dimen\z@{\ht\z@ \advance\dimen\z@ -\ht\tw@}%
2028   \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.

```

2029 \def\save@sf@q#1{\leavevmode
2030   \begingroup
2031   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2032   \endgroup}

```

3.20.1. 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.

```

2033 \ProvideTextCommand{\quotedblbase}{OT1}{%
2034   \save@sf@q{\set@low@box{\textquotedblright}/}%
2035   \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.

```

2036 \ProvideTextCommandDefault{\quotedblbase}{%
2037   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2038 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2039   \save@sf@q{\set@low@box{\textquoteright}/}%
2040   \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.

```
2041 \ProvideTextCommandDefault{\quotesinglbase}{%
2042   \UseTextSymbol{OT1}{\quotesinglbase}}
```

\guillemetleft

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

```
2043 \ProvideTextCommand{\guillemetleft}{OT1}{%
2044   \ifmmode
2045     \ll
2046   \else
2047     \save@sf@q{\nobreak
2048       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2049   \fi}
2050 \ProvideTextCommand{\guillemetright}{OT1}{%
2051   \ifmmode
2052     \gg
2053   \else
2054     \save@sf@q{\nobreak
2055       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2056   \fi}
2057 \ProvideTextCommand{\guillemotleft}{OT1}{%
2058   \ifmmode
2059     \ll
2060   \else
2061     \save@sf@q{\nobreak
2062       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2063   \fi}
2064 \ProvideTextCommand{\guillemotright}{OT1}{%
2065   \ifmmode
2066     \gg
2067   \else
2068     \save@sf@q{\nobreak
2069       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2070   \fi}
```

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

```
2071 \ProvideTextCommandDefault{\guillemetleft}{%
2072   \UseTextSymbol{OT1}{\guillemetleft}}
2073 \ProvideTextCommandDefault{\guillemetright}{%
2074   \UseTextSymbol{OT1}{\guillemetright}}
2075 \ProvideTextCommandDefault{\guillemotleft}{%
2076   \UseTextSymbol{OT1}{\guillemotleft}}
2077 \ProvideTextCommandDefault{\guillemotright}{%
2078   \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft

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

```
2079 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2080   \ifmmode
2081     <%
2082   \else
2083     \save@sf@q{\nobreak
2084       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2085   \fi}
2086 \ProvideTextCommand{\guilsinglright}{OT1}{%
2087   \ifmmode
2088     >%
2089   \else
2090     \save@sf@q{\nobreak
2091       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2092   \fi}
```

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

```
2093 \ProvideTextCommandDefault{\guilsinglleft}{%
2094 \UseTextSymbol{OT1}{\guilsinglleft}}
2095 \ProvideTextCommandDefault{\guilsinglright}{%
2096 \UseTextSymbol{OT1}{\guilsinglright}}
```

3.20.2. 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.

```
2097 \DeclareTextCommand{\ij}{OT1}{%
2098 i\kern-0.02em\bbl@allowhyphens j}
2099 \DeclareTextCommand{\IJ}{OT1}{%
2100 I\kern-0.02em\bbl@allowhyphens J}
2101 \DeclareTextCommand{\ij}{T1}{\char188}
2102 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2103 \ProvideTextCommandDefault{\ij}{%
2104 \UseTextSymbol{OT1}{\ij}}
2105 \ProvideTextCommandDefault{\IJ}{%
2106 \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).

```
2107 \def\crrtic@{\hrule height0.1ex width0.3em}
2108 \def\crrtic@{\hrule height0.1ex width0.33em}
2109 \def\ddj@{%
2110 \setbox0\hbox{d}\dimen@=\ht0
2111 \advance\dimen@lex
2112 \dimen@.45\dimen@
2113 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2114 \advance\dimen@ii.5ex
2115 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2116 \def\DDJ@{%
2117 \setbox0\hbox{D}\dimen@=.55\ht0
2118 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2119 \advance\dimen@ii.15ex % correction for the dash position
2120 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2121 \dimen\thr@{\expandafter\rem@pt\the\fontdimen7\font\dimen@
2122 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2123 %
2124 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2125 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2126 \ProvideTextCommandDefault{\dj}{%
2127 \UseTextSymbol{OT1}{\dj}}
2128 \ProvideTextCommandDefault{\DJ}{%
2129 \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.

```
2130 \DeclareTextCommand{\SS}{OT1}{SS}
2131 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

3.20.3. 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.

```
2132 \ProvideTextCommandDefault{\glq}{%
2133 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2134 \ProvideTextCommand{\grq}{T1}{%
2135 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}%
2136 \ProvideTextCommand{\grq}{TU}{%
2137 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2138 \ProvideTextCommand{\grq}{OT1}{%
2139 \save@sf@q{\kern-.0125em
2140 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2141 \kern.07em\relax}}
2142 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq`

`\grqq` The ‘german’ double quotes.

```
2143 \ProvideTextCommandDefault{\glqq}{%
2144 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2145 \ProvideTextCommand{\grqq}{T1}{%
2146 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2147 \ProvideTextCommand{\grqq}{TU}{%
2148 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2149 \ProvideTextCommand{\grqq}{OT1}{%
2150 \save@sf@q{\kern-.07em
2151 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2152 \kern.07em\relax}}
2153 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq`

`\frq` The ‘french’ single guillemets.

```
2154 \ProvideTextCommandDefault{\flq}{%
2155 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}%
2156 \ProvideTextCommandDefault{\frq}{%
2157 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq`

`\frqq` The ‘french’ double guillemets.

```
2158 \ProvideTextCommandDefault{\flqq}{%
2159 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2160 \ProvideTextCommandDefault{\frqq}{%
2161 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

3.20.4. 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).

```

2162 \def\umlauthigh{%
2163   \def\bbbl@umlauta##1{\leavevmode\bgroup%
2164     \accent\csname\fontencoding dqpos\endcsname
2165     ##1\bbbl@allowhyphens\egroup}%
2166   \let\bbbl@umlaute\bbbl@umlauta}
2167 \def\umlautlow{%
2168   \def\bbbl@umlauta{\protect\lower@umlaut}}
2169 \def\umlautelowlow{%
2170   \def\bbbl@umlaute{\protect\lower@umlaut}}
2171 \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.

```

2172 \expandafter\ifx\csname U@D\endcsname\relax
2173   \csname newdimen\endcsname U@D
2174 \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.

```

2175 \def\lower@umlaut#1{%
2176   \leavevmode\bgroup
2177   \U@D lex%
2178   {\setbox\z@\hbox{%
2179     \char\csname\fontencoding dqpos\endcsname}%
2180     \dimen@ -.45ex\advance\dimen@\ht\z@
2181     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2182   \accent\csname\fontencoding dqpos\endcsname
2183   \fontdimen5\font\U@D #1%
2184   \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bbbl@umlauta` or `\bbbl@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 `\bbbl@umlauta` and/or `\bbbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2185 \AtBeginDocument{%
2186   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbbl@umlauta{a}}%
2187   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbbl@umlaute{e}}%
2188   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbbl@umlaute{i}}%
2189   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbbl@umlaute{i}}%
2190   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbbl@umlauta{o}}%
2191   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbbl@umlauta{u}}%
2192   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbbl@umlauta{A}}%
2193   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbbl@umlaute{E}}%
2194   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbbl@umlaute{I}}%
2195   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbbl@umlauta{O}}%
2196   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbbl@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`.

```

2197 \ifx\l@english\undefined
2198   \chardef\l@english\z@
2199 \fi

```

```

2200 % The following is used to cancel rules in ini files (see Amharic).
2201 \ifx\l@unhyphenated\@undefined
2202   \newlanguage\l@unhyphenated
2203 \fi

```

3.21. Layout

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

```

2204 \bbl@trace{Bidi layout}
2205 \providecommand\IfBabelLayout[3]{#3}%

```

3.22. 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.

```

2206 \bbl@trace{Input engine specific macros}
2207 \ifcase\bbl@engine
2208   \input txtbabel.def
2209 \or
2210   \input luababel.def
2211 \or
2212   \input xebabel.def
2213 \fi
2214 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}{}
2215 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}{}
2216 \ifx\babelposthyphenation\@undefined
2217   \let\babelposthyphenation\babelprehyphenation
2218   \let\babelpatterns\babelprehyphenation
2219   \let\babelcharproperty\babelprehyphenation
2220 \fi

```

3.23. Creating and modifying languages

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

```

2221 \bbl@trace{Creating languages and reading ini files}
2222 \let\bbl@extend@ini\@gobble
2223 \newcommand\babelprovide[2][]{%
2224   \let\bbl@savelangname\language
2225   \edef\bbl@savelocaleid{\the\localeid}%
2226   \global\let\bbl@afterload\@empty
2227   % Set name and locale id
2228   \edef\language{#2}%
2229   \bbl@id@assign
2230   % Initialize keys
2231   \bbl@vforeach{captions,date,import,main,script,language,%
2232     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2233     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2234     Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2235     @import}%
2236     {\bbl@csarg\let{KVP@##1}\@nnil}%
2237   \global\let\bbl@release@transforms\@empty
2238   \global\let\bbl@release@casing\@empty
2239   \let\bbl@calendars\@empty
2240   \global\let\bbl@inidata\@empty
2241   \global\let\bbl@extend@ini\@gobble
2242   \global\let\bbl@included@inis\@empty
2243   \gdef\bbl@key@list{;}%
2244   \bbl@ifunset{\bbl@passto#2}%
2245     {\def\bbl@tempa{#1}}%
2246     {\bbl@exp{\def\\bbl@tempa{[\bbl@passto#2],\unexpanded{#1}}}}%

```

```

2247 \expandafter\bbk@forkv\expandafter{\bbk@tempa}{%
2248 \in@{/{}}{##1}% With /, (re)sets a value in the ini
2249 \ifin@
2250 \bbk@renewinikey##1\@{##2}%
2251 \else
2252 \bbk@csarg\ifx{KVP@##1}\@nnil\else
2253 \bbk@error{unknown-provide-key}{##1}{}}{%
2254 \fi
2255 \bbk@csarg\def{KVP@##1}{##2}%
2256 \fi}%
2257 \chardef\bbk@howloaded=% 0:none; 1:ldf without ini; 2:ini
2258 \bbk@ifunset{date#2}\z@{\bbk@ifunset{\bbk@llevel@#2}\@ne\tw@}%
2259 % == init ==
2260 \ifx\bbk@screset\@undefined
2261 \bbk@ldfinit
2262 \fi
2263 % ==
2264 % If there is no import (last wins), use @import (internal, there
2265 % must be just one). To consider any order (because
2266 % \PassOptionsToLocale).
2267 \ifx\bbk@KVP@import\@nnil
2268 \let\bbk@KVP@import\bbk@KVP@import
2269 \fi
2270 % == date (as option) ==
2271 % \ifx\bbk@KVP@date\@nnil\else
2272 % \fi
2273 % ==
2274 \let\bbk@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2275 \ifcase\bbk@howloaded
2276 \let\bbk@lbkflag\@empty % new
2277 \else
2278 \ifx\bbk@KVP@hyphenrules\@nnil\else
2279 \let\bbk@lbkflag\@empty
2280 \fi
2281 \ifx\bbk@KVP@import\@nnil\else
2282 \let\bbk@lbkflag\@empty
2283 \fi
2284 \fi
2285 % == import, captions ==
2286 \ifx\bbk@KVP@import\@nnil\else
2287 \bbk@exp{\bbk@ifblank{\bbk@KVP@import}}%
2288 {\ifx\bbk@initoload\relax
2289 \begingroup
2290 \def\BabelBeforeIni##1##2{\gdef\bbk@KVP@import{##1}\endinput}%
2291 \bbk@input@texini{#2}%
2292 \endgroup
2293 \else
2294 \xdef\bbk@KVP@import{\bbk@initoload}%
2295 \fi}%
2296 {}%
2297 \let\bbk@KVP@date\@empty
2298 \fi
2299 \let\bbk@KVP@captions@\bbk@KVP@captions
2300 \ifx\bbk@KVP@captions\@nnil
2301 \let\bbk@KVP@captions\bbk@KVP@import
2302 \fi
2303 % ==
2304 \ifx\bbk@KVP@transforms\@nnil\else
2305 \bbk@replace\bbk@KVP@transforms{ }{,}%
2306 \fi
2307 % Load ini
2308 % -----
2309 \ifcase\bbk@howloaded

```

```

2310 \bbl@provide@new{#2}%
2311 \else
2312 \bbl@ifblank{#1}%
2313 {}% With \bbl@load@basic below
2314 {\bbl@provide@renew{#2}}%
2315 \fi
2316 % Post tasks
2317 % -----
2318 % == subsequent calls after the first provide for a locale ==
2319 \ifx\bbl@inidata\@empty\else
2320 \bbl@extend@ini{#2}%
2321 \fi
2322 % ==
2323 \ifx\bbl@KVP@mapdot\@nnil\else
2324 \def\bbl@tempa{\@empty}%
2325 \ifx\bbl@KVP@mapdot\bbl@tempa\else
2326 \bbl@exp{\gdef\<bbl@map@.@\@language>{\[bbl@KVP@mapdot]}}%
2327 \fi
2328 \fi
2329 % == ensure captions ==
2330 \ifx\bbl@KVP@captions\@nnil\else
2331 \bbl@ifunset{bbl@extracaps@#2}%
2332 {\bbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2333 {\bbl@exp{\\babelensure[exclude=\\today,
2334 include=\[bbl@extracaps@#2]]{#2}}}%
2335 \let\bbl@tempc\language
2336 \bbl@replace\bbl@tempc{-}{@}%
2337 \bbl@ifunset{bbl@ensure@\bbl@tempc}%
2338 {\bbl@exp{%
2339 \\DeclareRobustCommand\<bbl@ensure@\bbl@tempc>[1]{%
2340 \\foreignlanguage{\language}%
2341 {###1}}}%
2342 }%
2343 \bbl@exp{%
2344 \\bbl@tglobal\<bbl@ensure@\bbl@tempc>%
2345 \\bbl@tglobal\<bbl@ensure@\bbl@tempc\space>}%
2346 \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.

```

2347 \bbl@load@basic{#2}%
2348 % == script, language ==
2349 % Override the values from ini or defines them
2350 \ifx\bbl@KVP@script\@nnil\else
2351 \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2352 \fi
2353 \ifx\bbl@KVP@language\@nnil\else
2354 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2355 \fi
2356 \ifcase\bbl@engine\or
2357 \bbl@ifunset{bbl@chrng@\language}{}%
2358 {\directlua{
2359 Babel.set_chranges_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2360 \fi
2361 % == Line breaking: intraspace, intrapenalty ==
2362 % For CJK, East Asian, Southeast Asian, if interspace in ini
2363 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2364 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2365 \fi
2366 \bbl@provide@intraspace
2367 % == Line breaking: justification ==
2368 \ifx\bbl@KVP@justification\@nnil\else

```

```

2369 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2370 \fi
2371 \ifx\bbl@KVP@linebreaking\@nnil\else
2372 \bbl@xin@{,\bbl@KVP@linebreaking,}%
2373 {,elongated,kashida,cjk,padding,unhyphenated,}%
2374 \ifin@
2375 \bbl@csarg\xdef
2376 {\lbrk@{\language\name}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2377 \fi
2378 \fi
2379 \bbl@xin@{/e}{/\bbl@cl{\lbrk}}}%
2380 \ifin@{\else\bbl@xin@{/k}{/\bbl@cl{\lbrk}}}\fi
2381 \ifin@\bbl@arabicjust\fi
2382 \bbl@xin@{/p}{/\bbl@cl{\lbrk}}}%
2383 \ifin@\AtBeginDocument{\@nameuse{\bbl@tibetanjust}}\fi
2384 % == Line breaking: hyphenate.other.(locale|script) ==
2385 \ifx\bbl@lbrkflag\@empty
2386 \bbl@ifunset{\bbl@hyotl@{\language\name}}{ }%
2387 {\bbl@csarg\bbl@replace{\hyotl@{\language\name}}{ }{,}%
2388 \bbl@startcommands*{\language\name}}%
2389 \bbl@csarg\bbl@foreach{\hyotl@{\language\name}}{
2390 \ifcase\bbl@engine
2391 \ifnum##1<257
2392 \SetHyphenMap{\BabelLower{##1}{##1}}%
2393 \fi
2394 \else
2395 \SetHyphenMap{\BabelLower{##1}{##1}}%
2396 \fi}%
2397 \bbl@endcommands}%
2398 \bbl@ifunset{\bbl@hyots@{\language\name}}{ }%
2399 {\bbl@csarg\bbl@replace{\hyots@{\language\name}}{ }{,}%
2400 \bbl@csarg\bbl@foreach{\hyots@{\language\name}}{
2401 \ifcase\bbl@engine
2402 \ifnum##1<257
2403 \global\lccode##1=##1\relax
2404 \fi
2405 \else
2406 \global\lccode##1=##1\relax
2407 \fi}}%
2408 \fi
2409 % == Counters: maparabic ==
2410 % Native digits, if provided in ini (TeX level, xe and lua)
2411 \ifcase\bbl@engine\else
2412 \bbl@ifunset{\bbl@dgnat@{\language\name}}{ }%
2413 {\expandafter\ifx\csname\bbl@dgnat@{\language\name}\endcsname\@empty\else
2414 \expandafter\expandafter\expandafter
2415 \bbl@setdigits\csname\bbl@dgnat@{\language\name}\endcsname
2416 \ifx\bbl@KVP@maparabic\@nnil\else
2417 \ifx\bbl@latinarabic\@undefined
2418 \expandafter\let\expandafter\@arabic
2419 \csname\bbl@counter@{\language\name}\endcsname
2420 \else % i.e., if layout=counters, which redefines \@arabic
2421 \expandafter\let\expandafter\bbl@latinarabic
2422 \csname\bbl@counter@{\language\name}\endcsname
2423 \fi
2424 \fi
2425 \fi}%
2426 \fi
2427 % == Counters: mapdigits ==
2428 % > luababel.def
2429 % == Counters: alph, Alph ==
2430 \ifx\bbl@KVP@alph\@nnil\else
2431 \bbl@exp{%

```

```

2432     \\bbl@add\<bbl@preextras@\language\>{%
2433     \\babel@save\\@alph
2434     \let\\@alph\<bbl@cntr@bbl@KVP@alph @\language\>}}%
2435 \fi
2436 \ifx\bbl@KVP@Alph\@nnil\else
2437     \bbl@exp{%
2438         \\bbl@add\<bbl@preextras@\language\>{%
2439         \\babel@save\\@Alph
2440         \let\\@Alph\<bbl@cntr@bbl@KVP@Alph @\language\>}}%
2441 \fi
2442 % == Counters: mapdot ==
2443 \ifx\bbl@KVP@mapdot\@nnil\else
2444     \bbl@foreach\bbl@list@the{%
2445         \bbl@ifunset{the##1}{}%
2446         {{\bbl@ncarg\let\bbl@tempd{the##1}%
2447         \bbl@carg\bbl@sreplace{the##1}{.}{\bbl@map@lbl{.}}%
2448         \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2449         \bbl@exp{\gdef\<the##1>{\[the##1]}}}%
2450         \fi}}}%
2451 \edef\bbl@tempb{enumi,enumii,enumiii,enumiv}%
2452 \bbl@foreach\bbl@tempb{%
2453     \bbl@ifunset{label##1}{}%
2454     {{\bbl@ncarg\let\bbl@tempd{label##1}%
2455     \bbl@carg\bbl@sreplace{label##1}{.}{\bbl@map@lbl{.}}%
2456     \expandafter\ifx\csname label##1\endcsname\bbl@tempd\else
2457     \bbl@exp{\gdef\<label##1>{\[label##1]}}}%
2458     \fi}}}%
2459 \fi
2460 % == Casing ==
2461 \bbl@release@casing
2462 \ifx\bbl@KVP@casing\@nnil\else
2463     \bbl@csarg\xdef{casing@\language\}%
2464     {\@nameuse{\bbl@casing@\language\}\bbl@maybextx\bbl@KVP@casing}%
2465 \fi
2466 % == Calendars ==
2467 \ifx\bbl@KVP@calendar\@nnil
2468     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}}%
2469 \fi
2470 \def\bbl@tempe##1 ##2\@{ % Get first calendar
2471     \def\bbl@tempa{##1}}%
2472     \bbl@exp{\\bbl@tempe\bbl@KVP@calendar\space\\@}%
2473 \def\bbl@tempe##1.##2.##3.##4\@{ %
2474     \def\bbl@tempf{##3}% First
2475     \let\bbl@tempc\@empty \let\bbl@tempb\@empty
2476     \bbl@cal@getconvert\bbl@tempc{##1}%
2477     \bbl@cal@getconvert\bbl@tempb{##2}}%
2478 \expandafter\bbl@tempe\bbl@tempa...\@
2479 \bbl@csarg\edef{calpr@\language\}{%
2480     \ifx\bbl@tempc\@empty\else
2481         calendar=\bbl@tempc
2482     \fi
2483     \ifx\bbl@tempb\@empty\else
2484         ,variant=\bbl@tempb
2485     \fi
2486     \ifx\bbl@tempf\@empty\else
2487         ,convert=\bbl@tempf
2488     \fi}%
2489 % == engine specific extensions ==
2490 % Defined in XXXbabel.def
2491 \bbl@provide@extra{#2}%
2492 % == require.babel in ini ==
2493 % To load or reload the babel-*.tex, if require.babel in ini
2494 \ifx\bbl@beforestart\relax\else % But not in doc aux or body

```

```

2495 \bbl@ifunset{\bbl@rqtex@\language\name}{}%
2496 {\expandafter\ifx\csname\bbl@rqtex@\language\name\endcsname\@empty\else
2497 \let\BabelBeforeIni\@gobbletwo
2498 \chardef\atcatcode=\catcode\@
2499 \catcode\@=11\relax
2500 \def\CurrentOption{#2}%
2501 \bbl@input@texini{\bbl@cs{rqtex@\language\name}}%
2502 \catcode\@=\atcatcode
2503 \let\atcatcode\relax
2504 \global\bbl@csarg\let{rqtex@\language\name}\relax
2505 \fi}%
2506 \bbl@foreach\bbl@calendars{%
2507 \bbl@ifunset{\bbl@ca-##1}{%
2508 \chardef\atcatcode=\catcode\@
2509 \catcode\@=11\relax
2510 \InputIfFileExists{babel-ca-##1.tex}{}}}%
2511 \catcode\@=\atcatcode
2512 \let\atcatcode\relax}%
2513 }%
2514 \fi
2515 % == frenchspacing ==
2516 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2517 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2518 \ifin@
2519 \bbl@extras@wrap{\bbl@pre@fs}%
2520 {\bbl@pre@fs}%
2521 {\bbl@post@fs}%
2522 \fi
2523 % == transforms ==
2524 % > luababel.def
2525 \def\CurrentOption{#2}%
2526 \@nameuse{\bbl@icsave@#2}%
2527 % == main ==
2528 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2529 \let\language\bbl@savelangname
2530 \chardef\localeid\bbl@savelocaleid\relax
2531 \fi
2532 % == ==
2533 \ifx\ldf@finish\@onlypreamble\else
2534 \bbl@afterload
2535 \fi
2536 % == hyphenrules (apply if current) ==
2537 \ifx\bbl@KVP@hyphenrules\@nnil\else
2538 \ifnum\bbl@savelocaleid=\localeid
2539 \language\@nameuse{l@\language\name}%
2540 \fi
2541 \fi}

```

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

```

2542 \def\bbl@provide@new#1{%
2543 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2544 \namedef{extras#1}{}%
2545 \namedef{noextras#1}{}%
2546 \bbl@startcommands*{#1}{captions}%
2547 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2548 \def\bbl@tempb##1{% elt for \bbl@captionslist
2549 \ifx##1\@nnil\else
2550 \bbl@exp{%
2551 \\\SetString\\##1{%
2552 \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2553 \expandafter\bbl@tempb
2554 \fi}%

```

```

2555     \expandafter\bbbl@tempb\bbbl@captionslist\@nnil
2556 \else
2557     \ifx\bbbl@initoload\relax
2558         \bbbl@read@ini{\bbbl@KVP@captions}2%    % Here letters cat = 11
2559     \else
2560         \bbbl@read@ini{\bbbl@initoload}2%        % Same
2561     \fi
2562 \fi
2563 \StartBabelCommands*{#1}{date}%
2564 \ifx\bbbl@KVP@date\@nnil
2565     \bbbl@exp{%
2566         \\SetString\\today{\\bbbl@nocaption{today}{#1today}}}%
2567 \else
2568     \bbbl@savetoday
2569     \bbbl@savedate
2570 \fi
2571 \bbbl@endcommands
2572 \bbbl@load@basic{#1}%
2573 % == hyphenmins == (only if new)
2574 \bbbl@exp{%
2575     \gdef\<#1hyphenmins>{%
2576         {\bbbl@ifunset{\bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2577         {\bbbl@ifunset{\bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}%
2578     % == hyphenrules (also in renew) ==
2579     \bbbl@provide@hyphens{#1}%
2580     % == main ==
2581     \ifx\bbbl@KVP@main\@nnil\else
2582         \expandafter\main@language\expandafter{#1}%
2583     \fi}
2584 %
2585 \def\bbbl@provide@renew#1{%
2586     \ifx\bbbl@KVP@captions\@nnil\else
2587         \StartBabelCommands*{#1}{captions}%
2588         \bbbl@read@ini{\bbbl@KVP@captions}2%    % Here all letters cat = 11
2589         \EndBabelCommands
2590     \fi
2591     \ifx\bbbl@KVP@date\@nnil\else
2592         \StartBabelCommands*{#1}{date}%
2593         \bbbl@savetoday
2594         \bbbl@savedate
2595         \EndBabelCommands
2596     \fi
2597     % == hyphenrules (also in new) ==
2598     \ifx\bbbl@lbfkflag\@empty
2599         \bbbl@provide@hyphens{#1}%
2600     \fi
2601     % == main ==
2602     \ifx\bbbl@KVP@main\@nnil\else
2603         \expandafter\main@language\expandafter{#1}%
2604     \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.

```

2605 \def\bbbl@load@basic#1{%
2606     \ifcase\bbbl@howloaded\or\or
2607         \ifcase\csname bbl@llevel@\language\endcsname
2608             \bbbl@csarg\let\lname@\language\relax
2609         \fi
2610     \fi
2611     \bbbl@ifunset{\bbbl@lname@#1}%
2612     {\def\BabelBeforeIni##1##2{%
2613         \begingroup

```

```

2614 \let\bbl@ini@captions@aux\@gobbletwo
2615 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{%
2616 \bbl@read@ini{##1}l%
2617 \ifx\bbl@initoload\relax\endinput\fi
2618 \endgroup}%
2619 \begingroup % boxed, to avoid extra spaces:
2620 \ifx\bbl@initoload\relax
2621 \bbl@input@texini{##1}%
2622 \else
2623 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2624 \fi
2625 \endgroup}%
2626 {}

```

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).

```

2627 \def\bbl@load@info#1{%
2628 \def\BabelBeforeIni##1##2{%
2629 \begingroup
2630 \bbl@read@ini{##1}0%
2631 \endinput % babel- .tex may contain onlypreamble's
2632 \endgroup}% boxed, to avoid extra spaces:
2633 {\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.

```

2634 \def\bbl@provide@hyphens#1{%
2635 \@tempcnta\m@ne % a flag
2636 \ifx\bbl@KVP@hyphenrules\@nnil\else
2637 \bbl@replace\bbl@KVP@hyphenrules{ },}%
2638 \bbl@foreach\bbl@KVP@hyphenrules{%
2639 \ifnum\@tempcnta=\m@ne % if not yet found
2640 \bbl@ifsamestring{##1}{+}%
2641 {\bbl@carg\addlanguage{l@##1}}%
2642 {}%
2643 \bbl@ifunset{l@##1}% After a possible +
2644 {}%
2645 {\@tempcnta\@nameuse{l@##1}}%
2646 \fi}%
2647 \ifnum\@tempcnta=\m@ne
2648 \bbl@warning{%
2649 Requested 'hyphenrules' for '\language' not found:\%
2650 \bbl@KVP@hyphenrules.\%
2651 Using the default value. Reported}%
2652 \fi
2653 \fi
2654 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2655 \ifx\bbl@KVP@captions@\@nnil
2656 \bbl@ifunset\bbl@hyphr{##1}{ }% use value in ini, if exists
2657 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@##1}}}%
2658 {}%
2659 {\bbl@ifunset{l@\bbl@ccl{hyphr}}}%
2660 {}% if hyphenrules found:
2661 {\@tempcnta\@nameuse{l@\bbl@ccl{hyphr}}}}%
2662 \fi
2663 \fi
2664 \bbl@ifunset{l@#1}%
2665 {\ifnum\@tempcnta=\m@ne
2666 \bbl@carg\adddialect{l@#1}\language
2667 \else
2668 \bbl@carg\adddialect{l@#1}\@tempcnta
2669 \fi}%

```

```

2670     {\ifnum\@tempcnta=\m@ne\else
2671       \global\bbbl@carg\chardef{l@#1}\@tempcnta
2672       \fi}}

```

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

```

2673 \def\bbbl@input@texini#1{%
2674   \bbbl@bsphack
2675   \bbbl@exp{%
2676     \catcode`\%%=14 \catcode`\%%=0
2677     \catcode`\%{=1 \catcode`\%}=2
2678     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2679     \catcode`\%%=\the\catcode`\%\relax
2680     \catcode`\%=\the\catcode`\%\relax
2681     \catcode`\%{=\the\catcode`\%\relax
2682     \catcode`\%}=\the\catcode`\%\relax}%
2683   \bbbl@esphack}

```

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

```

2684 \def\bbbl@iniline#1\bbbl@iniline{%
2685   \@ifnextchar[\bbbl@inisect{\@ifnextchar;\bbbl@iniskip\bbbl@inistore}#1\@@}% ]
2686 \def\bbbl@inisect[#1]#2\@@{\def\bbbl@section{#1}}
2687 \def\bbbl@iniskip#1\@@{%      if starts with ;
2688 \def\bbbl@inistore#1=#2\@@{%   full (default)
2689   \bbbl@trim@def\bbbl@tempa{#1}%
2690   \bbbl@trim\toks@{#2}%
2691   \bbbl@ifsamestring{\bbbl@tempa}{\@include}%
2692     {\bbbl@read@subini{\the\toks@}}%
2693     {\bbbl@xin@;\bbbl@section/\bbbl@tempa;}{\bbbl@key@list}%
2694     \ifin@else
2695       \bbbl@xin@{,identification/include.}%
2696       {\bbbl@section/\bbbl@tempa}%
2697       \ifin@\xdef\bbbl@included@inis{\the\toks@}\fi
2698       \bbbl@exp{%
2699         \\g@addto@macro\\bbbl@inidata{%
2700           \\bbbl@elt{\bbbl@section}{\bbbl@tempa}{\the\toks@}}}%
2701       \fi}}
2702 \def\bbbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbbl@read@ini)
2703   \bbbl@trim@def\bbbl@tempa{#1}%
2704   \bbbl@trim\toks@{#2}%
2705   \bbbl@xin@{.identification.}{.\bbbl@section.}%
2706   \ifin@
2707     \bbbl@exp{\\g@addto@macro\\bbbl@inidata{%
2708       \\bbbl@elt{identification}{\bbbl@tempa}{\the\toks@}}}%
2709   \fi}

```

3.24. Main loop in 'provide'

Now, the 'main loop', \bbbl@read@ini, which **must be executed inside a group**. At this point, \bbbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 (without import) or 2 (which import). The value -1 is used with \DocumentMetadata.

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

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

```

2710 \def\bbbl@loop@ini#1{%

```

```

2711 \loop
2712 \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2713 \endlinechar\m@ne
2714 \read#1 to \bbl@line
2715 \endlinechar`\^^M
2716 \ifx\bbl@line\@empty\else
2717 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2718 \fi
2719 \repeat}
2720 %
2721 \def\bbl@read@subini#1{%
2722 \ifx\bbl@readsubstream\@undefined
2723 \csname newread\endcsname\bbl@readsubstream
2724 \fi
2725 \openin\bbl@readsubstream=babel-#1.ini
2726 \ifeof\bbl@readsubstream
2727 \bbl@error{no-ini-file}{#1}{}}}%
2728 \else
2729 {\bbl@loop@ini\bbl@readsubstream}%
2730 \fi
2731 \closein\bbl@readsubstream}
2732 %
2733 \ifx\bbl@readstream\@undefined
2734 \csname newread\endcsname\bbl@readstream
2735 \fi
2736 \def\bbl@read@ini#1#2{%
2737 \global\let\bbl@extend@ini\@gobble
2738 \openin\bbl@readstream=babel-#1.ini
2739 \ifeof\bbl@readstream
2740 \bbl@error{no-ini-file}{#1}{}}}%
2741 \else
2742 % == Store ini data in \bbl@inidata ==
2743 \catcode`\ =10 \catcode`\ =12
2744 \catcode`\ [=12 \catcode`\ ]=12 \catcode`\ ==12 \catcode`\ &=12
2745 \catcode`\ ;=12 \catcode`\ |=12 \catcode`\ %=14 \catcode`\ -=12
2746 \ifnum#2=\m@ne % Just for the info
2747 \edef\language\tag\bbl@metalang}%
2748 \fi
2749 \ifnum#2=\m@ne
2750 \bbl@info{Metadata: '\bbl@metalang' requested, '#1' provided.\\%
2751 Fetching the locale name from babel-#1.ini\@gobble}%
2752 \else
2753 \bbl@info{Importing
2754 \ifcase#2font and identification \or basic \fi
2755 data for '\language'\%
2756 from babel-#1.ini. Reported}%
2757 \fi
2758 \ifnum#2<\@ne
2759 \global\let\bbl@inidata\@empty
2760 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2761 \fi
2762 \def\bbl@section{identification}%
2763 \bbl@exp{%
2764 \\bbl@inistore tag.ini=#1\\@@
2765 \\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\@@}%
2766 \bbl@loop@ini\bbl@readstream
2767 % == Process stored data ==
2768 \ifnum#2=\m@ne
2769 \def\bbl@tempa##1 ##2\@{##1}% Get first name
2770 \def\bbl@elt##1##2##3{%
2771 \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2772 {\edef\language{\bbl@tempa##3 \@}%
2773 \let\localname\language

```

```

2774         \bbl@id@assign
2775         \def\bbl@elt####1####2####3{}}%
2776     {}}%
2777     \bbl@inidata
2778     \fi
2779     \bbl@csarg\xdef{lini@\language}\bbl@inidata{#1}%
2780     \def\bbl@section{identification}%
2781     \bbl@exp{\\bbl@inistore name.declared=\language\\@@}%
2782     \bbl@read@ini@aux
2783     % == 'Export' data ==
2784     \bbl@ini@exports{#2}%
2785     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2786     \global\bbl@csarg\let{inidata@tag@\localeinfo*\language}\bbl@inidata
2787     \global\let\bbl@inidata\empty
2788     \bbl@xin@{,\language,},{,\bbl@ini@loaded,}%
2789     \ifin@else
2790         \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
2791         \bbl@tglobal\bbl@ini@loaded
2792     \fi
2793     \ifnum#2>\m@ne
2794         \@ifpackagewith{babel}{licr=unextended}{}%
2795         {\ifcase\bbl@engine\else % Find a better place
2796             \bbl@xin@{,\bbl@cl{sbcp},},{,Cyr,}%
2797             \ifin@
2798                 \g@addto@macro\bbl@afterload{\bbl@once{licr-cyr}}{\input{cyr2uni.def}}}%
2799             \fi
2800         \fi}%
2801     \fi
2802     \fi
2803     \closein\bbl@readstream}
2804 \def\bbl@read@ini@aux{%
2805     \let\bbl@savestrings\empty
2806     \let\bbl@savetoday\empty
2807     \let\bbl@savestate\empty
2808     \def\bbl@elt####1####2####3{%
2809         \def\bbl@section{##1}%
2810         \in@{=date.}{=##1}% Find a better place
2811         \ifin@
2812             \bbl@ifunset{bbl@inikv@##1}%
2813             {\bbl@ini@calendar{##1}}%
2814             {}%
2815         \fi
2816         \bbl@ifunset{bbl@inikv@##1}{}%
2817         {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2818     \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.

```

2819 \def\bbl@extend@ini@aux#1{%
2820     \bbl@startcommands*{#1}{captions}%
2821     % Activate captions/... and modify exports
2822     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2823         \setlocalecaption{#1}{##1}{##2}}%
2824     \def\bbl@inikv@captions##1##2{%
2825         \bbl@ini@captions@aux{##1}{##2}}%
2826     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2827     \def\bbl@exportkey##1##2##3{%
2828         \bbl@ifunset{bbl@kv@##2}{%
2829             {\expandafter\ifx\csname bbl@kv@##2\endcsname\empty\else
2830                 \bbl@exp{\global\let<bbl@##1@\language>\<bbl@kv@##2>}}%
2831             \fi}}%
2832     % As with \bbl@read@ini, but with some changes
2833     \bbl@read@ini@aux

```

```

2834 \bbl@ini@exports\tw@
2835 % Update inidata@lang by pretending the ini is read.
2836 \def\bbl@elt##1##2##3{%
2837   \def\bbl@section{##1}%
2838   \bbl@iniline##2=##3\bbl@iniline}%
2839 \csname bbl@inidata@#1\endcsname
2840 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2841 \StartBabelCommands*{#1}{date}% And from the import stuff
2842 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2843 \bbl@savetoday
2844 \bbl@savedate
2845 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2846 \def\bbl@ini@calendar#1{%
2847   \lowercase{\def\bbl@tempa{=#1=}}%
2848   \bbl@replace\bbl@tempa{=date.gregorian}{}}%
2849   \bbl@replace\bbl@tempa{=date.}{}}%
2850   \in@{.licr=}{#1=}%
2851   \ifin@
2852     \ifcase\bbl@engine
2853       \bbl@replace\bbl@tempa{.licr=}{}}%
2854     \else
2855       \let\bbl@tempa\relax
2856     \fi
2857   \fi
2858   \ifx\bbl@tempa\relax\else
2859     \bbl@replace\bbl@tempa{=}{}}%
2860     \ifx\bbl@tempa\@empty\else
2861       \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2862     \fi
2863     \bbl@exp{%
2864       \def\bbl@inikv@#1>####1####2{%
2865         \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2866     \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).

```

2867 \def\bbl@renewinikey#1/#2\@#3{%
2868   \global\let\bbl@extend@ini\bbl@extend@ini@aux
2869   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2870   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2871   \bbl@trim\toks@{#3}% value
2872   \bbl@exp{%
2873     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2874     \\gaddto@macro\\bbl@inidata{%
2875       \\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.

```

2876 \def\bbl@exportkey#1#2#3{%
2877   \bbl@ifunset{bbl@kv@#2}%
2878   {\bbl@csarg\gdef{#1@\language name}{#3}}%
2879   {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2880     \bbl@csarg\gdef{#1@\language name}{#3}%
2881   \else
2882     \bbl@exp{\global\let<bbl@#1@\language name>\<bbl@kv@#2>}%
2883   \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.

```
2884 \def\bbl@iniwarning#1{%
2885   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2886   {\bbl@warning{%
2887     From babel-\bbl@cs{lini@language}.ini:\\%
2888     \bbl@cs{@kv@identification.warning#1}\\%
2889     Reported}}}%
2890 %
2891 \let\bbl@release@transforms\@empty
2892 \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).

```
2893 \def\bbl@ini@exports#1{%
2894   % Identification always exported
2895   \bbl@iniwarning{}%
2896   \ifcase\bbl@engine
2897     \bbl@iniwarning{.pdfLaTeX}%
2898   \or
2899     \bbl@iniwarning{.luaLaTeX}%
2900   \or
2901     \bbl@iniwarning{.XeLaTeX}%
2902   \fi%
2903   \bbl@exportkey{lllevel}{identification.load.level}{}%
2904   \bbl@exportkey{elname}{identification.name.english}{}%
2905   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2906     {\csname bbl@elname@language\endcsname}}%
2907   \bbl@exportkey{tbcpr}{identification.tag.bcp47}{}%
2908   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2909   \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
2910   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2911   \bbl@exportkey{esname}{identification.script.name}{}%
2912   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2913     {\csname bbl@esname@language\endcsname}}%
2914   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2915   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2916   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2917   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2918   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2919   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2920   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2921   % Also maps bcp47 -> language
2922   \bbl@csarg\def{bcp@map@bbl@cl{tbcpr}}{\language}%
2923   \ifcase\bbl@engine\or
2924     \directlua{%
2925       Babel.locale_props[\the\bbl@cs{id@language}].script
2926       = '\bbl@cl{sbcpr}'}%
2927   \fi
2928   % Conditional
2929   \ifnum#1>\z@ % -1 or 0 = only info, 1 = basic, 2 = (re)new
2930     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2931     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2932     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2933     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
```

```

2934 \bbl@exportkey{rgthm}{typography.righthypenmin}{3}%
2935 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2936 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2937 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2938 \bbl@exportkey{intsp}{typography.intraspace}{}%
2939 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2940 \bbl@exportkey{chrng}{characters.ranges}{}%
2941 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2942 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2943 \ifnum#1=\tw@ % only (re)new
2944 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2945 \bbl@tglobal\bbl@savetoday
2946 \bbl@tglobal\bbl@savedate
2947 \bbl@savestrings
2948 \fi
2949 \fi}

```

3.25. Processing keys in ini

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

```

2950 \def\bbl@inikv#1#2{%      key=value
2951 \toks@{#2}%              This hides #'s from ini values
2952 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2953 \let\bbl@inikv@identification\bbl@inikv
2954 \let\bbl@inikv@date\bbl@inikv
2955 \let\bbl@inikv@typography\bbl@inikv
2956 \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.

```

2957 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\empty x-\fi}
2958 \def\bbl@inikv@characters#1#2{%
2959 \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2960 {\bbl@exp{%
2961 \\\g@addto@macro{\bbl@release@casing{%
2962 \\\bbl@casemapping}{\language}\unexpanded{#2}}}%
2963 {\in@{casing.}{#1}% e.g., casing.Uv = uV
2964 \ifin@
2965 \lowercase{\def\bbl@tempb{#1}%
2966 \bbl@replace\bbl@tempb{casing.}}}%
2967 \bbl@exp{\\\g@addto@macro{\bbl@release@casing{%
2968 \\\bbl@casemapping
2969 {\\\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2970 \else
2971 \bbl@inikv{#1}{#2}%
2972 \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’. The asterisk is a ‘terminator’.

```

2973 \def\bbl@inikv@counters#1#2{%
2974 \bbl@ifsamestring{#1}{digits}%
2975 {\bbl@error{digits-is-reserved}{}}}%
2976 {}%
2977 \def\bbl@tempc{#1}%
2978 \bbl@trim@def{\bbl@tempb*}{#2}%
2979 \in@{.1$}{#1$}%
2980 \ifin@
2981 \bbl@replace\bbl@tempc{.1}{}%
2982 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%

```

```

2983 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2984 \fi
2985 \in@{.F.}{#1}%
2986 \ifin@else\in@{.S.}{#1}\fi
2987 \ifin@
2988 \bbl@csarg\protected@xdef{cnt@#1@language}{\bbl@tempb*}%
2989 \else
2990 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2991 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2992 \bbl@csarg{\global\expandafter\let}{cnt@#1@language}\bbl@tempa
2993 \fi
2994 \in@{.D$}{#1$}%
2995 \ifin@else\in@{.C$}{#1$}\fi
2996 \ifin@
2997 \bbl@replace\bbl@tempc{.D}{}%
2998 \bbl@replace\bbl@tempc{.C}{}%
2999 \bbl@exp{\gdef\<bbl@cnt@bbl@tempc @language>####1{%
3000 \ \expandafter\ \bbl@zhnumeral\ \expandafter{\ \number####1}{\bbl@tempc}}}%
3001 \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.

```

3002 \ifcase\bbl@engine
3003 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3004 \bbl@ini@captions@aux{#1}{#2}}
3005 \else
3006 \def\bbl@inikv@captions#1#2{%
3007 \bbl@ini@captions@aux{#1}{#2}}
3008 \fi

```

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

```

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

```

```

3041 \bbl@exp{%
3042   \\bbl@add\\bbl@savestrings{%
3043     \\SetString<\bbl@tempa name>{\the\toks@}}}%
3044 \toks@ \expandafter{\bbl@captionslist}%
3045 \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3046 \ifin@ \else
3047   \bbl@exp{%
3048     \\bbl@add<\bbl@extracaps@ \language name>{\<\bbl@tempa name>}%
3049     \\bbl@to global \<\bbl@extracaps@ \language name>}%
3050 \fi
3051 \fi}

```

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

```

3052 \def\bbl@list@the{%
3053   part,chapter,section,subsection,subsubsection,paragraph,%
3054   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3055   table,page,footnote,mpfootnote,mpfn}
3056 %
3057 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3058   \bbl@ifunset{\bbl@map@#1@ \language name}%
3059     {\@nameuse{#1}}%
3060     {\@nameuse{\bbl@map@#1@ \language name}}}
3061 %
3062 \def\bbl@map@lbl#1{% #1:a sign, eg, .
3063   \ifin@csname#1\else
3064     \bbl@ifunset{\bbl@map@#1@ \language name}%
3065       {#1}%
3066       {\@nameuse{\bbl@map@#1@ \language name}}}
3067 \fi}
3068 %
3069 \def\bbl@inikv@labels#1#2{%
3070   \in@{.map}{#1}%
3071   \ifin@
3072     \in@{,dot.map,}{, #1,}%
3073   \ifin@
3074     \global\@namedef{\bbl@map@. @ \language name}{#2}%
3075   \fi
3076   \ifx\bbl@KVP@labels\@nnil\else
3077     \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3078     \ifin@
3079       \def\bbl@tempc{#1}%
3080       \bbl@replace\bbl@tempc{.map}{}%
3081       \in@{, #2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3082       \bbl@exp{%
3083         \gdef\<\bbl@map@\bbl@tempc @ \language name>%
3084         {\ifin@ \<#2> \else \\localecounter{#2} \fi}}%
3085       \bbl@foreach\bbl@list@the{%
3086         \bbl@ifunset{the##1}{}%
3087         {\bbl@ncarg\let\bbl@tempd{the##1}%
3088          \bbl@exp{%
3089            \\bbl@sreplace\<the##1>%
3090            {\<\bbl@tempc>{##1}}%
3091            {\\bbl@map@cnt{\bbl@tempc}{##1}}%
3092            \\bbl@sreplace\<the##1>%
3093            {\<\@empty @\bbl@tempc>\<c@##1>%
3094             {\\bbl@map@cnt{\bbl@tempc}{##1}}%
3095             \\bbl@sreplace\<the##1>%
3096             {\\csname @\bbl@tempc\\endcsname\<c@##1>%
3097              {\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3098             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3099               \bbl@exp{\gdef\<the##1>{\[the##1]}}%
3100             \fi}}%
3101       \fi

```

```

3102 \fi
3103 %
3104 \else
3105 % The following code is still under study. You can test it and make
3106 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3107 % language dependent.
3108 \in@{enumerate.}{#1}%
3109 \ifin@
3110 \def\bbl@tempa{#1}%
3111 \bbl@replace\bbl@tempa{enumerate.}{}%
3112 \def\bbl@toreplace{#2}%
3113 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3114 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3115 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3116 \toks@{\expandafter\bbl@toreplace}%
3117 \bbl@exp{%
3118 \\\bbl@add<extras\language>{%
3119 \\\babel@save<labelenum\romannumeral\bbl@tempa>%
3120 \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3121 \\\bbl@tglobal<extras\language>}%
3122 \fi
3123 \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.

```

3124 \def\bbl@chapttype{chapter}
3125 \ifx\@makechapterhead\@undefined
3126 \let\bbl@patchchapter\relax
3127 \else\ifx\thechapter\@undefined
3128 \let\bbl@patchchapter\relax
3129 \else\ifx\ps@headings\@undefined
3130 \let\bbl@patchchapter\relax
3131 \else
3132 \def\bbl@patchchapter{%
3133 \global\let\bbl@patchchapter\relax
3134 \gdef\bbl@chfmt{%
3135 \bbl@ifunset\bbl@chapttype fmt@\language}%
3136 {\@chapapp\space\thechapter}%
3137 {\@nameuse\bbl@chapttype fmt@\language}}}%
3138 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3139 \bbl@sreplace\ps@headings{\@chapapp\thechapter}{\bbl@chfmt}%
3140 \bbl@sreplace\chaptermark{\@chapapp\thechapter}{\bbl@chfmt}%
3141 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3142 \bbl@tglobal\appendix
3143 \bbl@tglobal\ps@headings
3144 \bbl@tglobal\chaptermark
3145 \bbl@tglobal\@makechapterhead}
3146 \let\bbl@patchappendix\bbl@patchchapter
3147 \fi\fi\fi
3148 \ifx\@part\@undefined
3149 \let\bbl@patchpart\relax
3150 \else
3151 \def\bbl@patchpart{%
3152 \global\let\bbl@patchpart\relax
3153 \gdef\bbl@partformat{%
3154 \bbl@ifunset\bbl@partfmt@\language}%
3155 {\@partname\nobreakspace\thepart}%
3156 {\@nameuse\bbl@partfmt@\language}}}%
3157 \bbl@sreplace\@part{\@partname\nobreakspace\thepart}{\bbl@partformat}%
3158 \bbl@tglobal\@part}
3159 \fi

```

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

```

3160 \def\bbl@cal@getconvert#1#2{% Used in provide
3161   \in{:}{#2}%
3162   \ifin@def\bbl@tempf{#2}\else\def#1{#2}\fi
3163 \let\bbl@calendar@empty
3164 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3165 \def\bbl@localedate#1#2#3#4{%
3166   \begingroup
3167     \edef\bbl@they{#2}%
3168     \edef\bbl@them{#3}%
3169     \edef\bbl@thed{#4}%
3170     \edef\bbl@tempe{%
3171       \bbl@ifunset\bbl@calpr{\language\language}\bbl@cl{calpr}},%
3172       #1}%
3173     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3174     \let\bbl@ld@calendar@empty
3175     \let\bbl@ld@variant@empty
3176     \let\bbl@ld@convert@relax
3177     \let\bbl@ld@@convert@relax
3178     \expandafter\bbl@forkv\expandafter{\bbl@tempe}{\bbl@csarg\edef{ld##1}{##2}}%
3179     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3180     \ifx\bbl@ld@convert@relax
3181       \let\bbl@ld@convert\bbl@ld@@convert
3182     \fi
3183     \ifx\bbl@ld@convert@empty
3184       \ifx\bbl@ld@calendar@empty
3185         \let\bbl@ld@convert@relax
3186       \else
3187         \let\bbl@ld@convert\bbl@ld@calendar
3188       \fi
3189     \fi
3190     \ifx\bbl@ld@convert@relax\else
3191       \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3192       {\bbl@ld@convert}\bbl@they\bbl@them\bbl@thed
3193     \fi
3194     \@nameuse{\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3195     \edef\bbl@calendar{% Used in \month..., too
3196       \bbl@ld@calendar
3197       \ifx\bbl@ld@variant@empty\else
3198         .\bbl@ld@variant
3199       \fi}%
3200     \bbl@cased
3201     {\@nameuse{\bbl@date@\language @\bbl@calendar}%
3202      \bbl@they\bbl@them\bbl@thed}%
3203   \endgroup}
3204 %
3205 \def\bbl@printdate#1{%
3206   \ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}}
3207 \def\bbl@printdate@i#1[#2]#3#4#5{%
3208   \bbl@usedategrouptrue
3209   \def\bbl@tempc{#1}%
3210   \bbl@replace\bbl@tempc{-}{@}%
3211   \@nameuse{\bbl@ensure@\bbl@tempc}{\localedate[#2][#3][#4][#5]}
3212 %
3213 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3214 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3215   \bbl@trim@def\bbl@tempa{#1.#2}%
3216   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3217   {\bbl@trim@def\bbl@tempa{#3}%
3218    \bbl@trim\toks@{#5}%
3219    \@temptokena\expandafter{\bbl@savedate}%
3220    \bbl@exp{% Reverse order - in ini last wins

```

```

3221 \def\\bbl@savedate{%
3222   \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3223   \the\@temptokena}}}%
3224 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3225 {\lowercase{\def\bbl@tempb{#6}}}%
3226 \bbl@trim@def\bbl@toreplace{#5}%
3227 \bbl@TG@date
3228 \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3229 \ifx\bbl@savetoday\empty
3230 \bbl@exp{%
3231   \\AfterBabelCommands{%
3232     \gdef\<\language name date>{\\protect\<\language name date >}%
3233     \gdef\<\language name date >{\\bbl@printdate{\language name}}}%
3234   \def\\bbl@savetoday{%
3235     \\SetString\\today{%
3236       \<\language name date>[@convert]%
3237       {\the\year}{\the\month}{\the\day}}}%
3238   \fi}%
3239 {}}}

```

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).

```

3240 \let\bbl@calendar\empty
3241 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3242   \bbl@xin@{\$calendrica:}{\$#2}%
3243   \ifin@
3244     \directlua{Babel.calendrica = Babel.calendrica or require("calendrica")}%
3245     \edef\bbl@tempa{\$#2}%
3246     \bbl@replace\bbl@tempa{\$calendrica:}{}%
3247     \bbl@afterelse
3248     \bbl@exp{\\bbl@calendrica#1\\@@{\bbl@tempa}}%
3249   \else
3250     \bbl@afterfi
3251     \@nameuse{\bbl@ca@#2}#1\@@
3252   \fi}
3253 \def\bbl@calendrica#1-#2-#3\@@#4#5#6#7{%
3254   \directlua{
3255     local r = Babel.calendrica.date(
3256       {year=\number#1, month=\number#2, day=\number#3,
3257       'gregorian'}:to('#4')
3258     token.set_macro('\bbl@stripslash#5', tostring(r.year))
3259     token.set_macro('\bbl@stripslash#6', tostring(r.month))
3260     token.set_macro('\bbl@stripslash#7', tostring(r.day))
3261   }}
3262 \newcommand\babelDateSpace{\nobreakspace}
3263 \newcommand\babelDateDot{.\@}
3264 \newcommand\babelDated[1]{\number#1}
3265 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3266 \newcommand\babelDateM[1]{\number#1}
3267 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3268 \newcommand\babelDateMMMM[1]{%
3269   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3270 \newcommand\babelDatey[1]{\number#1}%
3271 \newcommand\babelDateyy[1]{%
3272   \ifnum#1<10 0\number#1 %
3273   \else\ifnum#1<100 \number#1 %
3274   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3275   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3276   \else
3277     \bbl@error{limit-two-digits}{}}}%

```

```

3278 \fi\fi\fi\fi}}
3279 \newcommand\BabelDateyyy[1]{\number#1}}
3280 \newcommand\BabelDateU[1]{\number#1}}%
3281 \def\bbl@replace@finish@iii#1{%
3282 \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3283 \def\bbl@TG@date{%
3284 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3285 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3286 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3287 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecncr[###1]}%
3288 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3289 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyy{###1}}%
3290 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3291 \bbl@replace\bbl@toreplace{[M]}{\bbl@datecncr[###2]}%
3292 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3293 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3294 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3295 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecncr[###3]}%
3296 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3297 \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3298 \bbl@replace\bbl@toreplace{[U]}{\bbl@datecncr[###1]}%
3299 \bbl@replace@finish@iii\bbl@toreplace}
3300 \def\bbl@datecncr{\expandafter\bbl@xdatecncr\expandafter}
3301 \def\bbl@xdatecncr[#1#2]{\localenumeral{#2}{#1}}

```

3.26. French spacing (again)

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

```

3302 \AddToHook{begindocument/before}{%
3303 \let\bbl@normalsf\normalsfcodes
3304 \let\normalsfcodes\relax}
3305 \AtBeginDocument{%
3306 \ifx\bbl@normalsf\empty
3307 \ifnum\sfcodes\@m
3308 \let\normalsfcodes\frenchspacing
3309 \else
3310 \let\normalsfcodes\nonfrenchspacing
3311 \fi
3312 \else
3313 \let\normalsfcodes\bbl@normalsf
3314 \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 \bbl@transforms@aux ... \relax, and stores them in \bbl@release@transforms. However, since building a list enclosed in braces isn't trivial, the replacements are added after a comma, and then \bbl@transforms@aux adds the braces.

```

3315 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3316 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3317 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3318 #1[#2]{#3}{#4}{#5}}
3319 \begingroup
3320 \catcode`\%=12
3321 \catcode`\&=14
3322 \gdef\bbl@transforms#1#2#3{\&
3323 \directlua{
3324 local str = [=[#2]=]
3325 str = str:gsub('%.%d+%.%d+$', '')
3326 token.set_macro('babeltempa', str)
3327 }&
3328 \def\babeltempc{&

```

```

3329 \bbl@xin@{\babeltempa,}{,\bbl@KVP@transforms,}&%
3330 \ifin@else
3331 \bbl@xin@{\babeltempa,}{,\bbl@KVP@transforms,}&%
3332 \fi
3333 \ifin@
3334 \bbl@foreach\bbl@KVP@transforms{&%
3335 \bbl@xin@{\babeltempa,}{,##1,}&%
3336 \ifin@ &% font:font:transform syntax
3337 \directlua{
3338 local t = {}
3339 for m in string.gmatch('##1'..' ':'', '(.)') do
3340 table.insert(t, m)
3341 end
3342 table.remove(t)
3343 token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3344 }&%
3345 \fi}&%
3346 \in@{.0$}{#2$}&%
3347 \ifin@
3348 \directlua{&% (\attribute) syntax
3349 local str = string.match([\bbl@KVP@transforms],
3350 '%([^(%[-])^(%))]-\babeltempa')
3351 if str == nil then
3352 token.set_macro('babeltempb', '')
3353 else
3354 token.set_macro('babeltempb', ',attribute=' .. str)
3355 end
3356 }&%
3357 \toks@{#3}&%
3358 \bbl@exp{&%
3359 \\g@addto@macro\\bbl@release@transforms{&%
3360 \relax &% Closes previous \bbl@transforms@aux
3361 \\bbl@transforms@aux
3362 \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3363 {\language\the\toks@}}&%
3364 \else
3365 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3366 \fi
3367 \fi}
3368 \endgroup

```

3.27. 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.

```

3369 \def\bbl@provide@lsys#1{%
3370 \bbl@ifunset\bbl@lname@#1{%
3371 {\bbl@load@info{#1}}%
3372 }%
3373 \bbl@csarg\let{lsys@#1}\@empty
3374 \bbl@ifunset\bbl@sname@#1{\bbl@csarg\gdef{sname@#1}{Default}}{%
3375 \bbl@ifunset\bbl@sotf@#1{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3376 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3377 \bbl@ifunset\bbl@lname@#1{%
3378 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}%
3379 \ifcase\bbl@engine\or\or
3380 \bbl@ifunset\bbl@prehc@#1{%
3381 {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3382 }%
3383 {\ifx\bbl@xenohyph\undefined
3384 \global\let\bbl@xenohyph\bbl@xenohyph@

```

```

3385 \ifx\AtBeginDocument\@notprerr
3386 \expandafter\@secondoftwo % to execute right now
3387 \fi
3388 \AtBeginDocument{%
3389 \bbl@patchfont{\bbl@xenohyph}%
3390 {\expandafter\select@language\expandafter{\language name}}}%
3391 \fi}%
3392 \fi
3393 \bbl@csarg\bbl@tglobal{\lsys#1}}

```

A tool to define the macros for native digits from the list provided in the `ini` file. Somewhat convoluted because there are 10 digits, but only 9 arguments in $\text{T}_{\text{E}}\text{X}$. Non-digits characters are kept. The first macro is the generic “localized” command.

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

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

```

3434 \newcommand\localenumberal[2]{%
3435   \bbl@ifunset{bbl@cntr@#1@\language\language}%
3436   {#2}%
3437   {\bbl@cs{cntr@#1@\language\language}{#2}}}
3438 \def\bbl@localecctr#1#2{\localenumberal{#2}{#1}}
3439 \newcommand\localecounter[2]{%
3440   \expandafter\bbl@localecctr
3441   \expandafter{\number\csname c@#2\endcsname}{#1}}
3442 \def\bbl@alphnumer#1#2{%
3443   \expandafter\bbl@alphnumer@i\number#2 76543210\@@{#1}}
3444 \def\bbl@alphnumer@i#1#2#3#4#5#6#7#8\@@#9{%
3445   \ifcase\@car#8\@nil\or   % Currently <10000, but prepared for bigger
3446   \bbl@alphnumer@ii{#9}000000#1\or
3447   \bbl@alphnumer@ii{#9}000000#1#2\or
3448   \bbl@alphnumer@ii{#9}000000#1#2#3\or
3449   \bbl@alphnumer@ii{#9}000000#1#2#3#4\else
3450   \bbl@error{counter-too-large}{>9999}{}}%
3451   \fi}
3452 \def\bbl@alphnumer@ii#1#2#3#4#5#6#7#8{%
3453   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language\language}%
3454   {\bbl@cs{cntr@#1.4@\language\language}{#5}%
3455    \bbl@cs{cntr@#1.3@\language\language}{#6}%
3456    \bbl@cs{cntr@#1.2@\language\language}{#7}%
3457    \bbl@cs{cntr@#1.1@\language\language}{#8}%
3458    \ifnum#6#7#8>\z@
3459     \bbl@ifunset{bbl@cntr@#1.S.321@\language\language}{}}%
3460     {\bbl@cs{cntr@#1.S.321@\language\language}}}%
3461   \fi}%
3462   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language\language}}}}

```

Some Chinese counters follow special rules.

```

3463 \def\bbl@zhnumer#1#2{%
3464   \ifnum#1<10 \bbl@zhnum000#1{#2}%
3465   \else\ifnum#1<100 \bbl@zhnum00#1{#2}%
3466   \else\ifnum#1<1000 \bbl@zhnum0#1{#2}%
3467   \else\ifnum#1<10000 \bbl@zhnum#1{#2}%
3468   \else\bbl@error{counter-too-large}{#1}{}}%
3469   \fi\fi\fi\fi}
3470 \def\bbl@zhnum#1#2#3#4#5{%
3471   \ifnum\numexpr#1#2#3#4\relax=\z@
3472     \localenumberal{#5.C}\@ne
3473   \else
3474     \ifnum#1>\z@\localenumberal{#5.D}{#1}\localenumberal{#5.C}5\fi % Thousands
3475     \ifnum#2>\z@\localenumberal{#5.D}{#2}\localenumberal{#5.C}4\fi % Hundreds
3476     \else\ifnum#1>\z@\ifnum#3#4>\z@\localenumberal{#5.C}\@ne\fi
3477     \fi\fi
3478     \ifnum#3>\z@ % Tens. Omit for 10-19.
3479     \ifnum#1#2=\z@\ifnum#3=\@ne\else\localenumberal{#5.D}{#3}\fi
3480     \else\localenumberal{#5.D}{#3}%
3481     \fi
3482     \localenumberal{#5.C}\thr@@
3483   \else
3484     \ifnum#2>\z@\ifnum#4>\z@\localenumberal{#5.C}\@ne\fi\fi
3485     \fi
3486     \ifnum#4>\z@\localenumberal{#5.D}{#4}\fi % Units
3487     \fi}

```

3.29. Casing

```

3488 \newcommand\BabelUppercaseMapping[3]{%
3489   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3490 \newcommand\BabelTitlecaseMapping[3]{%
3491   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

```

```

3492 \newcommand\BabelLowercaseMapping[3]{%
3493   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

    The parser for casing and casing. (variant).
3494 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3495   \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3496 \else
3497   \def\bbl@uftocode#1{\expandafter\string#1}
3498 \fi
3499 \def\bbl@casemapping#1#2#3{% 1:variant
3500   \def\bbl@tempa##1 ##2{% Loop
3501     \bbl@casemapping@i{##1}%
3502     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3503   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3504   \def\bbl@tempe{0}% Mode (upper/lower...)
3505   \def\bbl@tempc{#3}% Casing list
3506   \expandafter\bbl@tempa\bbl@tempc\@empty}
3507 \def\bbl@casemapping@i#1{%
3508   \def\bbl@tempb{#1}%
3509   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3510     \@nameuse{regex_replace_all:nnN}%
3511     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\{\0}\}\bbl@tempb
3512   \else
3513     \@nameuse{regex_replace_all:nnN}{.}{\{\0}\}\bbl@tempb
3514   \fi
3515   \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3516 \def\bbl@casemapping@ii#1#2#3\@@{%
3517   \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3518   \ifin@
3519     \edef\bbl@tempe{%
3520       \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3521   \else
3522     \ifcase\bbl@tempe\relax
3523       \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3524       \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3525     \or
3526       \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3527     \or
3528       \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3529     \or
3530       \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3531   \fi
3532 \fi}

```

3.30. Getting info

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

```

3533 \def\bbl@localeinfo#1#2{%
3534   \bbl@ifunset{bbl@info@#2}{#1}%
3535   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3536    {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3537 \newcommand\localeinfo[1]{%
3538   \ifx*#1\@empty
3539     \bbl@afterelse\bbl@localeinfo{}%
3540   \else
3541     \bbl@localeinfo
3542     {\bbl@error{no-ini-info}}{\{\}\{\}\}%
3543     {#1}%
3544   \fi}
3545 % \@namedef{bbl@info@name.locale}{lcname}
3546 \@namedef{bbl@info@tag.ini}{lini}
3547 \@namedef{bbl@info@name.english}{elname}
3548 \@namedef{bbl@info@name.opentype}{lname}

```

```

3549 \@namedef{bbl@info@tag.bcp47}{tbc}
3550 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3551 \@namedef{bbl@info@tag.opentype}{lotf}
3552 \@namedef{bbl@info@script.name}{esname}
3553 \@namedef{bbl@info@script.name.opentype}{sname}
3554 \@namedef{bbl@info@script.tag.bcp47}{sbc}
3555 \@namedef{bbl@info@script.tag.opentype}{sotf}
3556 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3557 \@namedef{bbl@info@variant.tag.bcp47}{vbc}
3558 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3559 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3560 \@namedef{bbl@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.

```

3561 ((*More package options)) ▶
3562 \DeclareOption{ensureinfo=off}{}
3563 (/*More package options)
3564 \let\BabelEnsureInfo\relax

```

More general, but non-expandable, is \getLocaleproperty.

```

3565 \newcommand\getLocaleproperty{%
3566   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3567 \def\bbl@getproperty@s#1#2#3{%
3568   \let#1\relax
3569   \def\bbl@elt##1##2##3{%
3570     \bbl@ifsamestring{##1/##2}{#3}%
3571     {\providecommand#1{##3}%
3572     \def\bbl@elt###1####2####3{}}}%
3573   {}}%
3574   \bbl@cs{inidata@#2}}%
3575 \def\bbl@getproperty@x#1#2#3{%
3576   \bbl@getproperty@s{#1}{#2}{#3}%
3577   \ifx#1\relax
3578     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3579   \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.

```

3580 \let\bbl@ini@loaded\@empty
3581 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3582 \def\ShowLocaleProperties#1{%
3583   \typeout{}}%
3584   \typeout{*** Properties for language '#1' ***}%
3585   \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3586   \@nameuse{bbl@inidata@#1}%
3587   \typeout{*****}}

```

3.31. 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 autoloader.bcp47 enabled *and* lazy loading, we must first build a name for the language, with the help of autoloader.bcp47.prefix. Then we use \provideprovide passing the options set with autoloader.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.

```

3588 \newif\ifbbl@bcpallowed
3589 \bbl@bcpallowedfalse
3590 \def\bbl@autoloader@options{@import}
3591 \def\bbl@provide@locale{%
3592   \ifx\babelprovide\undefined
3593     \bbl@error{base-on-the-fly}{}{}{}%

```

```

3594 \fi
3595 \let\bbl@auxname\language
3596 \ifbbl@bcptoname
3597   \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
3598   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
3599   \let\locale\language}%
3600 \fi
3601 \ifbbl@bcpallowed
3602   \expandafter\ifx\csname date\language\endcsname\relax
3603     \expandafter
3604     \bbl@bcplookup{\language}%
3605     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3606       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3607       \let\locale\language
3608       \expandafter\ifx\csname date\language\endcsname\relax
3609         \let\bbl@initload\bbl@bcp
3610         \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3611         \let\bbl@initload\relax
3612       \fi
3613       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\locale}%
3614     \fi
3615   \fi
3616 \fi
3617 \expandafter\ifx\csname date\language\endcsname\relax
3618   \IfFileExists{babel-\language.tex}%
3619   {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3620   {}%
3621 \fi}

```

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

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

```

3622 \providecommand\BCPdata{}
3623 \ifx\renewcommand\@undefined\else
3624   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3625   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3626     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3627     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3628     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3629   \def\bbl@bcpdata@ii#1#2{%
3630     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3631     {\bbl@error{unknown-ini-field}{#1}{}}}%
3632     {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3633     {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3634 \fi
3635 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3636 \@namedef{bbl@info@tag.tag.bcp47}{tbcpl} % For \BCPdata

```

4. Adjusting the Babel behavior

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

```

3637 \newcommand\babeladjust[1]{%
3638   \bbl@forkv{#1}{%
3639     \bbl@ifunset{bbl@ADJ@##1@##2}%
3640     {\bbl@cs{ADJ@##1}{##2}}%
3641     {\bbl@cs{ADJ@##1@##2}}}%
3642 %
3643 \def\bbl@adjust@lua#1#2{%
3644   \ifvmode
3645     \ifnum\currentgrouplevel=\z@

```

```

3646 \directlua{ Babel.#2 }%
3647 \expandafter\expandafter\expandafter\@gobble
3648 \fi
3649 \fi
3650 {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3651 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3652 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3653 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3654 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3655 \@namedef{bbl@ADJ@bidi.text@on}{%
3656 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3657 \@namedef{bbl@ADJ@bidi.text@off}{%
3658 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3659 \@namedef{bbl@ADJ@bidi.math@on}{%
3660 \let\bbl@noamsmath\@empty}
3661 \@namedef{bbl@ADJ@bidi.math@off}{%
3662 \let\bbl@noamsmath\relax}
3663 %
3664 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3665 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3666 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3667 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3668 %
3669 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3670 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3671 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3672 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3673 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3674 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3675 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3676 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3677 \@namedef{bbl@ADJ@justify.arabic@on}{%
3678 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3679 \@namedef{bbl@ADJ@justify.arabic@off}{%
3680 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3681 %
3682 \def\bbl@adjust@layout#1{%
3683 \ifvmode
3684 #1%
3685 \expandafter\@gobble
3686 \fi
3687 {\bbl@error{layout-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3688 \@namedef{bbl@ADJ@layout.tabular@on}{%
3689 \ifnum\bbl@tabular@mode=\tw@
3690 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3691 \else
3692 \chardef\bbl@tabular@mode\@ne
3693 \fi}
3694 \@namedef{bbl@ADJ@layout.tabular@off}{%
3695 \ifnum\bbl@tabular@mode=\tw@
3696 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3697 \else
3698 \chardef\bbl@tabular@mode\@z@
3699 \fi}
3700 \@namedef{bbl@ADJ@layout.lists@on}{%
3701 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3702 \@namedef{bbl@ADJ@layout.lists@off}{%
3703 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3704 %
3705 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3706 \bbl@bcpallowedtrue}
3707 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3708 \bbl@bcpallowedfalse}

```

```

3709 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3710   \def\bbl@bcp@prefix{#1}}
3711 \def\bbl@bcp@prefix{bcp47-}
3712 \@namedef{bbl@ADJ@autoload.options}#1{%
3713   \def\bbl@autoload@options{#1}}
3714 \def\bbl@autoload@bcptoptions{import}
3715 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3716   \def\bbl@autoload@bcptoptions{#1}}
3717 \newif\ifbbl@bcptoname
3718 %
3719 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3720   \bbl@bcptonametrue}
3721 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3722   \bbl@bcptonamefalse}
3723 %
3724 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3725   \directlua{ Babel.ignore_pre_char = function(node)
3726     return (node.lang == \the\csname l@nohyphenation\endcsname)
3727   end }}
3728 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3729   \directlua{ Babel.ignore_pre_char = function(node)
3730     return false
3731   end }}
3732 %
3733 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3734   \def\bbl@ignoreinterchar{%
3735     \ifnum\language=\l@nohyphenation
3736       \expandafter\@gobble
3737     \else
3738       \expandafter\@firstofone
3739     \fi}}
3740 \@namedef{bbl@ADJ@interchar.disable@off}{%
3741   \let\bbl@ignoreinterchar\@firstofone}
3742 %
3743 \@namedef{bbl@ADJ@select.write@shift}{%
3744   \let\bbl@restorelastskip\relax
3745   \def\bbl@savelastskip{%
3746     \let\bbl@restorelastskip\relax
3747     \ifvmode
3748       \ifdim\lastskip=\z@
3749         \let\bbl@restorelastskip\nobreak
3750       \else
3751         \bbl@exp{%
3752           \def\\bbl@restorelastskip{%
3753             \skip@=\the\lastskip
3754             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3755         \fi
3756       \fi}}
3757 \@namedef{bbl@ADJ@select.write@keep}{%
3758   \let\bbl@restorelastskip\relax
3759   \let\bbl@savelastskip\relax}
3760 \@namedef{bbl@ADJ@select.write@omit}{%
3761   \AddBabelHook{babel-select}{beforestart}{%
3762     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3763 \let\bbl@restorelastskip\relax
3764 \def\bbl@savelastskip##1\bbl@restorelastskip{}
3765 \@namedef{bbl@ADJ@select.encoding@off}{%
3766   \let\bbl@encoding@select@off\@empty}

```

4.1. Cross referencing macros

The $\text{\LaTeX}$ book states:

The key argument is any sequence of letters, digits, and punctuation symbols; upper-

and lowercase letters are regarded as different.

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

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

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

```
3767 <<(*More package options)>> ▶
3768 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3769 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3770 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3771 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3772 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3773 <</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.

```
3774 \bbl@trace{Cross referencing macros}
3775 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3776   \def\@newl@bel#1#2#3{%
3777     {\@safe@activetrue
3778       \bbl@ifunset{#1@#2}%
3779         \relax
3780         {\gdef\@multiplelabels{%
3781           \@latex@warning@no@line{There were multiply-defined labels}}%
3782           \@latex@warning@no@line{Label `#2' multiply defined}}%
3783       \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.

```
3784 \CheckCommand*\@testdef[3]{%
3785   \def\reserved@a{#3}%
3786   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3787   \else
3788     \@tempswatrue
3789   \fi}
```

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

```
3790 \def\@testdef#1#2#3{%
3791   \@safe@activetrue
3792   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3793   \def\bbl@tempb{#3}%
3794   \@safe@activesfalse
3795   \ifx\bbl@tempa\relax
3796   \else
3797     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3798   \fi
3799   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3800   \ifx\bbl@tempa\bbl@tempb
3801   \else
3802     \@tempswatrue
3803   \fi}
3804 \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.

```

3805 \bbl@xin@{R}\bbl@opt@safe
3806 \ifin@
3807 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3808 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3809 {\expandafter\strip@prefix\meaning\ref}%
3810 \ifin@
3811 \bbl@redefine\@kernel@ref#1{%
3812   \@safe@activetrue\org@@kernel@ref{#1}\@safe@activfalse}
3813 \bbl@redefine\@kernel@pageref#1{%
3814   \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3815 \bbl@redefine\@kernel@sref#1{%
3816   \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3817 \bbl@redefine\@kernel@spageref#1{%
3818   \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3819 \else
3820 \bbl@redefineroobust\ref#1{%
3821   \@safe@activetrue\org@ref{#1}\@safe@activfalse}
3822 \bbl@redefineroobust\pageref#1{%
3823   \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
3824 \fi
3825 \else
3826 \let\org@ref\ref
3827 \let\org@pageref\pageref
3828 \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.

```

3829 \bbl@xin@{B}\bbl@opt@safe
3830 \ifin@
3831 \bbl@redefine\@citex[#1]#2{%
3832   \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activfalse
3833   \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.)

```

3834 \AtBeginDocument{%
3835   \@ifpackageloaded{natbib}{%
3836     \def\@citex[#1][#2]#3{%
3837       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activfalse
3838       \org@@citex[#1][#2]{\bbl@tempa}}%
3839   }}

```

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

```

3840 \AtBeginDocument{%
3841   \@ifpackageloaded{cite}{%
3842     \def\@citex[#1]#2{%
3843       \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3844   }}

```

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

```
3845 \bbl@redefine\nocite#1{%
3846   \@safe@activestrue\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@activestrue is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order to determine during aux file processing which definition of \bibcite is needed we define \bibcite in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

```
3847 \bbl@redefine\bibcite{%
3848   \bbl@cite@choice
3849   \bibcite}
```

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

```
3850 \def\bbl@bibcite#1#2{%
3851   \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.

```
3852 \def\bbl@cite@choice{%
3853   \global\let\bibcite\bbl@bibcite
3854   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3855   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3856   \global\let\bbl@cite@choice\relax}
```

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

```
3857 \AtBeginDocument{\bbl@cite@choice}
```

\@bibitem One of the two internal L^AT_EX macros called by \bibitem that write the citation label on the aux file.

```
3858 \bbl@redefine\@bibitem#1{%
3859   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3860 \else
3861   \let\org@nocite\nocite
3862   \let\org@@citex\@citex
3863   \let\org@bibcite\bibcite
3864   \let\org@@bibitem\@bibitem
3865 \fi
```

4.2. Layout

```
3866 \newcommand\BabelPatchSection[1]{%
3867   \@ifundefined{#1}{}{%
3868     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3869     \@namedef{#1}{%
3870       \@ifstar{\bbl@presec@#1}}%
3871       {\@dblarg{\bbl@presec@x{#1}}}}}%
3872 \def\bbl@presec@x#1[#2]#3{%
3873   \bbl@exp{%
3874     \\select@language@x{\bbl@main@language}%
3875     \\bbl@cs{sspre@#1}%
3876     \\bbl@cs{ss@#1}%
3877     [\\foreignlanguage{\language@}{\unexpanded{#2}}}%
3878     {\\foreignlanguage{\language@}{\unexpanded{#3}}}%
3879     \ifcase\bbl@bidimode\else\\babelsubl{r}\fi}%
}
```

```

3880   \\select@language@x{\language@name}}
3881 \def\bbl@presec@#1#2{%
3882   \bbl@exp{%
3883     \\select@language@x{\bbl@main@language}%
3884     \\bbl@cs{sspre@#1}%
3885     \\bbl@cs{ss@#1}*%
3886     {\\foreignlanguage{\language@name}{\unexpanded{#2}}}%
3887     \ifcase\bbl@bidimode\else\\babelsublr{}\fi}%
3888   \\select@language@x{\language@name}}
3889 %
3890 \IfBabelLayout{sectioning}%
3891   {\BabelPatchSection{part}%
3892    \BabelPatchSection{chapter}%
3893    \BabelPatchSection{section}%
3894    \BabelPatchSection{subsection}%
3895    \BabelPatchSection{subsubsection}%
3896    \BabelPatchSection{paragraph}%
3897    \BabelPatchSection{subparagraph}%
3898    \def\babel@toc#1{%
3899      \select@language@x{\bbl@main@language}}}{%
3900 \IfBabelLayout{captions}%
3901   {\BabelPatchSection{caption}}}{%

```

\BabelFootnote Footnotes.

```

3902 \bbl@trace{Footnotes}
3903 \def\bbl@footnote#1#2#3{%
3904   \@ifnextchar[%
3905     {\bbl@footnote@o{#1}{#2}{#3}}%
3906     {\bbl@footnote@x{#1}{#2}{#3}}}%
3907 \long\def\bbl@footnote@x#1#2#3#4{%
3908   \begingroup
3909     \select@language@x{\bbl@main@language}%
3910     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3911   \endgroup}
3912 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3913   \begingroup
3914     \select@language@x{\bbl@main@language}%
3915     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3916   \endgroup}
3917 \def\bbl@footnotetext#1#2#3{%
3918   \@ifnextchar[%
3919     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3920     {\bbl@footnotetext@x{#1}{#2}{#3}}}%
3921 \long\def\bbl@footnotetext@x#1#2#3#4{%
3922   \begingroup
3923     \select@language@x{\bbl@main@language}%
3924     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3925   \endgroup}
3926 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3927   \begingroup
3928     \select@language@x{\bbl@main@language}%
3929     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3930   \endgroup}
3931 \def\BabelFootnote#1#2#3#4{%
3932   \ifx\bbl@fn@footnote\undefined
3933     \let\bbl@fn@footnote\footnote
3934   \fi
3935   \ifx\bbl@fn@footnotetext\undefined
3936     \let\bbl@fn@footnotetext\footnotetext
3937   \fi
3938   \bbl@ifblank{#2}%
3939     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3940      \@namedef{\bbl@stripslash#1text}%

```

```

3941      {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3942      {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}}%
3943      \@namedef{\bbl@stripslash#1text}%
3944      {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3945 \IfBabelLayout{footnotes}%
3946   {\ifcase\bbl@engine\or
3947     \AddToHook{begindocument/end}{%
3948       \ExplSyntaxOn
3949       \bbl@sreplace{\footnotetext{\color@begingroup}%
3950         {\color@begingroup\ifnum\bbl@insidemath>\z\localerestoredirs\fi}%
3951       \ExplSyntaxOff}%
3952   \fi
3953   \let\bbl@OL@footnote\footnote
3954   \BabelFootnote\footnote\language\language}%
3955   \BabelFootnote\localfootnote\language\language}%
3956   \BabelFootnote\mainfootnote{}{}%
3957   {}

```

4.3. 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.

```

3958 \bbl@trace{Marks}
3959 \IfBabelLayout{sectioning}
3960   {\ifx\bbl@opt@headfoot\@nnil
3961     \g@addto@macro\@resetactivechars{%
3962       \set@typeset@protect
3963       \expandafter\select@language\expandafter{\bbl@main@language}%
3964       \let\protect\noexpand
3965       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3966         \edef\thepage{%
3967           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3968       \fi}%
3969   \fi}
3970 {\ifbbl@single\else
3971   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3972     \markright#1{%
3973       \bbl@ifblank{#1}%
3974       {\org@markright{}}}%
3975     {\toks@{#1}%
3976       \bbl@exp{%
3977         \org@markright{\protect\foreignlanguage{\language}%
3978           {\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.)

```

3979   \ifx\@mkboth\markboth
3980     \def\bbl@tempc{\let\@mkboth\markboth}%
3981   \else
3982     \def\bbl@tempc{}%
3983   \fi
3984   \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3985     \markboth#1#2{%

```

```

3986 \protected@edef\bbl@tempb##1{%
3987 \protect\foreignlanguage
3988 {\language\name}{\protect\bbl@restore@actives##1}}%
3989 \bbl@ifblank{#1}%
3990 {\toks@{}}%
3991 {\toks@\expandafter{\bbl@tempb{#1}}}%
3992 \bbl@ifblank{#2}%
3993 {\@temptokena{}}%
3994 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3995 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3996 \bbl@tempc
3997 \fi} % end ifbbl@single, end \IfBabelLayout

```

4.4. Other packages

4.4.1. 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.

```

3998 \bbl@trace{Preventing clashes with other packages}
3999 \ifx\org@ref\undefined\else
4000 \bbl@xin@{R}\bbl@opt@safe
4001 \ifin@
4002 \AtBeginDocument{%
4003 \ifpackageloaded{ifthen}{%
4004 \bbl@redefine@long\ifthenelse#1#2#3{%
4005 \let\bbl@temp@pref\pageref
4006 \let\pageref\org@pageref
4007 \let\bbl@temp@ref\ref
4008 \let\ref\org@ref
4009 \@safe@activestrue
4010 \org@ifthenelse{#1}%
4011 {\let\pageref\bbl@temp@pref
4012 \let\ref\bbl@temp@ref
4013 \@safe@activesfalse
4014 #2}%
4015 {\let\pageref\bbl@temp@pref
4016 \let\ref\bbl@temp@ref
4017 \@safe@activesfalse
4018 #3}%
4019 }%
4020 }}%
4021 }
4022 \fi

```

4.4.2. 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`.

```

4023 \AtBeginDocument{%
4024   \ifpackageloaded{varioref}{%
4025     \bbl@redefine\@@vpageref#1[#2]#3{%
4026       \@safe@activetrue
4027       \org@@vpageref{#1}[#2]{#3}%
4028       \@safe@activesfalse}%
4029     \bbl@redefine\vrefpagenum#1#2{%
4030       \@safe@activetrue
4031       \org@vrefpagenum{#1}{#2}%
4032       \@safe@activesfalse}%

```

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

```

4033   \expandafter\def\csname Ref \endcsname#1{%
4034     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
4035   }{}%
4036 }
4037 \fi

```

4.4.3. `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.

```

4038 \AtEndOfPackage{%
4039   \AtBeginDocument{%
4040     \@ifpackageloaded{hhline}%
4041     {\expandafter\ifx\csname normal@char\string\endcsname\relax
4042       \else
4043         \makeatletter
4044         \def\@currname{hhline}\input{hhline.sty}\makeatother
4045         \fi}%
4046     {}%

```

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

```

4047 \def\substitutefontfamily#1#2#3{%
4048   \lowercase{\immediate\openout15=#1#2.fd\relax}%
4049   \immediate\write15{%
4050     \string\ProvidesFile{#1#2.fd}%
4051     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
4052     \space generated font description file]^J
4053     \string\DeclareFontFamily{#1}{#2}{^^J
4054     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
4055     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
4056     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
4057     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
4058     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
4059     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
4060     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
4061     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
4062     }%
4063   \closeout15

```

```

4064 }
4065 \@onlypreamble\substitutefontfamily

```

4.5. Encoding and fonts

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

\ensureascii

```

4066 \bbl@trace{Encoding and fonts}
4067 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
4068 \newcommand\BabelNonText{TS1,T3,TS3}
4069 \let\org@TeX\TeX
4070 \let\org@LaTeX\LaTeX
4071 \let\ensureascii\@firstofone
4072 \let\asciientcoding\@empty
4073 \AtBeginDocument{%
4074   \def\elt#1{,#1,}%
4075   \edef\bbl@tempa{\expandafter\gobbletwo\fontenc@load@list}%
4076   \let\elt\relax
4077   \let\bbl@tempb\@empty
4078   \def\bbl@tempc{OT1}%
4079   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
4080     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
4081   \bbl@foreach\bbl@tempa{%
4082     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
4083     \ifin@
4084       \def\bbl@tempb{#1}% Store last non-ascii
4085     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
4086     \ifin@else
4087       \def\bbl@tempc{#1}% Store last ascii
4088     \fi
4089   \fi}%
4090   \ifx\bbl@tempb\@empty\else
4091     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
4092     \ifin@else
4093       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
4094     \fi
4095     \let\asciientcoding\bbl@tempc
4096     \renewcommand\ensureascii[1]{%
4097       {\fontencoding{\asciientcoding}\selectfont#1}}%
4098     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
4099     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
4100   \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.

```

4101 \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.

```

4102 \AtBeginDocument{%
4103   \@ifpackageloaded{fontspec}%
4104     {\xdef\latinencoding{%

```

```

4105     \ifx\UTFencname\undefined
4106       EU\ifcase\bbl@engine\or2\or1\fi
4107     \else
4108       \UTFencname
4109     \fi}}%
4110 {\gdef\latinencoding{OT1}%
4111  \ifx\cf@encoding\bbl@t@one
4112    \xdef\latinencoding{\bbl@t@one}%
4113  \else
4114    \def\@elt#1{,#1,}%
4115    \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4116    \let\@elt\relax
4117    \bbl@xin@{,T1,}\bbl@tempa
4118    \ifin@
4119      \xdef\latinencoding{\bbl@t@one}%
4120    \fi
4121  \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.

```

4122 \DeclareRobustCommand{\latintext}{%
4123   \fontencoding{\latinencoding}\selectfont
4124   \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.

```

4125 \ifx\@undefined\DeclareTextFontCommand
4126   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
4127 \else
4128   \DeclareTextFontCommand{\textlatin}{\latintext}
4129 \fi

```

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

```

4130 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

4.6. 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 $\text{\TeX}$ grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\text{\TeX}$ -ja` shows, vertical typesetting is possible, too.

```

4131 \bbl@trace{Loading basic (internal) bidi support}
4132 \ifodd\bbl@engine
4133 \else % Any xe+lua bidi

```

```

4134 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
4135   \bbl@error{bidi-only-lua}{}}{}%
4136   \let\bbl@beforeforeign\leavevmode
4137   \AtEndOfPackage{%
4138     \EnableBabelHook{babel-bidi}%
4139     \bbl@xebidipar}
4140 \fi\fi
4141 \def\bbl@loadxebidi#1{%
4142   \ifx\RTLfootnotetext\undefined
4143     \AtEndOfPackage{%
4144       \EnableBabelHook{babel-bidi}%
4145       \ifx\fontspec\undefined
4146         \usepackage{fontspec}% bidi needs fontspec
4147       \fi
4148       \usepackage#1{bidi}%
4149       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4150       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4151         \ifnum\@nameuse\bbl@wdir\language=\tw@ % 'AL' bidi
4152           \bbl@digitsdotdash % So ignore in 'R' bidi
4153         \fi}}%
4154   \fi}
4155 \ifnum\bbl@bidimode>200 % Any xe bidi=
4156   \ifcase\expandafter\gobbletwo\the\bbl@bidimode\or
4157     \bbl@tentative{bidi=bidi}
4158     \bbl@loadxebidi{}
4159   \or
4160     \bbl@loadxebidi{[rldocument]}
4161   \or
4162     \bbl@loadxebidi{}
4163   \fi
4164 \fi
4165 \fi
4166 \ifnum\bbl@bidimode=\@ne % bidi=default
4167   \let\bbl@beforeforeign\leavevmode
4168   \ifodd\bbl@engine % lua
4169     \newattribute\bbl@attr@dir
4170     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4171     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4172   \fi
4173   \AtEndOfPackage{%
4174     \EnableBabelHook{babel-bidi}% pdf/lua/xe
4175     \ifodd\bbl@engine\else % pdf/xe
4176       \bbl@xebidipar
4177     \fi}
4178 \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.

```

4179 \bbl@trace{Macros to switch the text direction}
4180 \def\bbl@alscripts{%
4181   ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
4182 \def\bbl@rscripts{%
4183   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4184   Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
4185   Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
4186   Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
4187   Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
4188   Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
4189   Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
4190   Meroitic,N'Ko,Orkhon,Todhri}
4191 %
4192 \def\bbl@provide@dirs#1{%

```

```

4193 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4194 \ifin@
4195 \global\bbl@csarg\chardef{wdir@#1}\@ne
4196 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4197 \ifin@
4198 \global\bbl@csarg\chardef{wdir@#1}\tw@
4199 \fi
4200 \else
4201 \global\bbl@csarg\chardef{wdir@#1}\z@
4202 \fi
4203 \ifodd\bbl@engine
4204 \bbl@csarg\ifcase{wdir@#1}%
4205 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4206 \or
4207 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4208 \or
4209 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4210 \fi
4211 \fi}
4212 %
4213 \def\bbl@switchdir{%
4214 \bbl@ifunset{\bbl@sys@\language name}{\bbl@provide@sys@\language name}}{}%
4215 \bbl@ifunset{\bbl@wdir@\language name}{\bbl@provide@dirs@\language name}}{}%
4216 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
4217 \def\bbl@setdirs#1{%
4218 \ifcase\bbl@select@type
4219 \bbl@bodydir{#1}%
4220 \bbl@pardir{#1}% <- Must precede \bbl@texdir
4221 \fi
4222 \bbl@texdir{#1}}
4223 \ifnum\bbl@bidimode>\z@
4224 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4225 \DisableBabelHook{babel-bidi}
4226 \fi

```

Now the engine-dependent macros.

```

4227 \ifodd\bbl@engine % luatex=1
4228 \else % pdftex=0, xetex=2
4229 \newcount\bbl@dirlevel
4230 \chardef\bbl@thetexdir\z@
4231 \chardef\bbl@thepardir\z@
4232 \def\bbl@texdir#1{%
4233 \ifcase#1\relax
4234 \chardef\bbl@thetexdir\z@
4235 \@nameuse{setlatin}%
4236 \bbl@texdir@i\beginL\endL
4237 \else
4238 \chardef\bbl@thetexdir\@ne
4239 \@nameuse{setnonlatin}%
4240 \bbl@texdir@i\beginR\endR
4241 \fi}
4242 \def\bbl@texdir@i#1#2{%
4243 \ifhmode
4244 \ifnum\currentgrouplevel>\z@
4245 \ifnum\currentgrouplevel=\bbl@dirlevel
4246 \bbl@error{multiple-bidi}{\bbl@texdir@i}{\bbl@texdir@i}%
4247 \bgroup\aftergroup#2\aftergroup\egroup
4248 \else
4249 \ifcase\currentgrouptype\or % 0 bottom
4250 \aftergroup#2% 1 simple {}
4251 \or
4252 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4253 \or

```

```

4254      \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4255      \or\or\or % vbox vtop align
4256      \or
4257      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4258      \or\or\or\or\or\or % output math disc insert vcent mathchoice
4259      \or
4260      \aftergroup#2% 14 \beginngroup
4261      \else
4262      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4263      \fi
4264      \fi
4265      \bbl@dirlevel\currentgrouplevel
4266      \fi
4267      #1%
4268      \fi}
4269      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4270      \let\bbl@bodydir\@gobble
4271      \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).

```

4272      \def\bbl@xebidipar{%
4273      \let\bbl@xebidipar\relax
4274      \TeXeTstate\@ne
4275      \def\bbl@xeeverypar{%
4276      \ifcase\bbl@thepardir
4277      \ifcase\bbl@thetextdir\else\beginR\fi
4278      \else
4279      {\setbox\z@\lastbox\beginR\box\z@}%
4280      \fi}%
4281      \AddToHook{para/begin}{\bbl@xeeverypar}}
4282      \ifnum\bbl@bidimode>200 % Any xe bidi=
4283      \let\bbl@textdir@i\@gobbletwo
4284      \let\bbl@xebidipar\@empty
4285      \AddBabelHook{bidi}{foreign}{%
4286      \ifcase\bbl@thetextdir
4287      \BabelWrapText{\LR{##1}}}%
4288      \else
4289      \BabelWrapText{\RL{##1}}}%
4290      \fi}
4291      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4292      \fi
4293      \fi

```

A tool for weak L (mainly digits). We also disable warnings with `hyperref`. There is a variant for `bidi=unibidi`, but using formatting characters is discouraged by Unicode UAX9, so it must be revised.

```

4294      \providecommand\bbl@sublr[1]{\bbl@textdir\z@#1}
4295      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@sublr{##1}}}
4296      \AtBeginDocument{%
4297      \ifx\pdfstringdefDisableCommands\undefined\else
4298      \ifx\pdfstringdefDisableCommands\relax\else
4299      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4300      \fi
4301      \fi}

```

4.7. 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`.

```

4302 \bbl@trace{Local Language Configuration}
4303 \ifx\loadlocalcfg\@undefined
4304   \ifpackagewith{babel}{noconfigs}%
4305     {\let\loadlocalcfg\@gobble}%
4306     {\def\loadlocalcfg#1{%
4307       \InputIfFileExists{#1.cfg}%
4308       {\typeout{*****^J%
4309                 * Local config file #1.cfg used^^J%
4310                 *}}%
4311       \@empty}}
4312 \fi

```

4.8. 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).

```

4313 \bbl@trace{Language options}
4314 \def\BabelDefinitionFile#1#2#3{}
4315 \let\bbl@afterlang\relax
4316 \let\BabelModifiers\relax
4317 \let\bbl@loaded\@empty
4318 \def\bbl@load@language#1{%
4319   \InputIfFileExists{#1.ldf}%
4320   {\edef\bbl@loaded{\CurrentOption
4321     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4322     \expandafter\let\expandafter\bbl@afterlang
4323       \csname\CurrentOption.ldf-h@k\endcsname
4324     \expandafter\let\expandafter\BabelModifiers
4325       \csname bbl@mod@\CurrentOption\endcsname
4326     \bbl@exp{\AtBeginDocument{%
4327       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4328     {\bbl@error{unknown-package-option}{}}{}}

```

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.

Tagging PDF Span elements requires horizontal mode. With `DocumentMetada` we also force it with `\foreignlanguage` (this is also done in `bidi` texts).

```

4329 \ifx\GetDocumentProperties\@undefined\else
4330   \let\bbl@beforeforeign\leavevmode
4331   \edef\bbl@tempa{\GetDocumentProperties{document/other-languages}}%
4332   \let\bbl@collect@other\@empty
4333   \bbl@foreach\bbl@tempa{%
4334     \edef\bbl@metalang{#1}%
4335     \ifx\bbl@metalang\@empty\else
4336       \begingroup
4337         \expandafter
4338         \bbl@bcpllookup{\bbl@metalang}%
4339         \ifx\bbl@bcp\relax
4340           \ifx\bbl@opt@main\@nnil
4341             \bbl@error{no-locale-for-meta}{\bbl@metalang}{}{}%
4342           \fi
4343         \else
4344           \bbl@read@ini{\bbl@bcp}\m@ne

```

```

4345         \xdef\bbl@collect@other{\bbl@collect@other,\language}%
4346         \GenericInfo{}\{(babel) \@spaces\@spaces\@spaces
4347         Passing '\language' to babel\@gobble}%
4348     \fi
4349 \endgroup
4350 \fi}
4351 \ifx\bbl@collect@other\@empty\else
4352     \xdef\bbl@language@opts{\bbl@collect@other,\bbl@language@opts}%
4353 \fi
4354 \edef\bbl@metalang{\GetDocumentProperties{document/language}}%
4355 \ifx\bbl@metalang\@empty\else
4356     \begingroup
4357         \expandafter
4358         \bbl@bcplookup{\bbl@metalang}%
4359         \ifx\bbl@bcp\relax
4360             \ifx\bbl@opt@main\@nnil
4361                 \bbl@error{no-locale-for-meta}{\bbl@metalang}{\}%
4362             \fi
4363         \else
4364             \bbl@read@ini{\bbl@bcp}\m@ne
4365             \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4366             \ifx\bbl@opt@main\@nnil
4367                 \global\let\bbl@opt@main\language
4368             \fi
4369             \GenericInfo{}\{(babel) \@spaces\@spaces\@spaces
4370             Passing '\language' to babel\@gobble}%
4371         \fi
4372     \endgroup
4373 \fi
4374 \fi
4375 \ifx\bbl@opt@config\@nnil
4376     \ifpackagewith{babel}{noconfigs}{\}%
4377     {\InputIfFileExists{bblopts.cfg}%
4378     {\bbl@info{Configuration files are deprecated, as\\%
4379         they can break document portability.\\%
4380         Reported}%
4381     \typeout{*****^J%
4382         * Local config file bblopts.cfg used^^J%
4383         *}}%
4384     }{\}%
4385 \else
4386     \InputIfFileExists{\bbl@opt@config.cfg}%
4387     {\bbl@info{Configuration files are deprecated, as\\%
4388         they can break document portability.\\%
4389         Reported}%
4390     \typeout{*****^J%
4391         * Local config file \bbl@opt@config.cfg used^^J%
4392         *}}%
4393     {\bbl@error{config-not-found}{\}{\}}%
4394 \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 available locales, and which version (ldf or ini) will be loaded. This is done by first loading the corresponding `babel-⟨name⟩.tex` file.

The second argument of `\BabelBeforeIni` may content a `\BabelDefinitionFile` which defines `\bbl@tempa` and `\bbl@tempb` and saves the third argument for the moment of the actual loading. If there is no `\BabelDefinitionFile` the last element is usually empty, and the ini file is loaded. The values are used to build a list in the form ‘main-or-not’ / ‘ldf-or-ldfini-flag’ // ‘option-name’ // ‘bcp-tag’ / ‘ldf-name-or-none’. The ‘main-or-not’ element is 0 by default and set to 10 later if necessary (by prepending 1). The ‘bcp-tag’ is stored here so that the corresponding ini file can be be

loaded directly (with @import).

```

4395 \def\BabelBeforeIni#1#2{%
4396   \def\bbl@tempa{\@m}% <- Default if no \BDefFile
4397   \let\bbl@tempb\@empty
4398   #2%
4399   \edef\bbl@toload{%
4400     \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4401     \bbl@toload@last}%
4402   \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4403 \def\BabelDefinitionFile#1#2#3{%
4404   \def\bbl@tempa{#1}\def\bbl@tempb{#2}%
4405   \@namedef{bbl@preldf@CurrentOption}{#3}%
4406   \endinput}%

```

For efficiency, first preprocess the class options to remove those with =, which are becoming increasingly frequent (no language should contain this character). Here we use the more robust macro to traverse a clist from the $\TeX$ layer.

```

4407 \def\bbl@tempf{,}
4408 \@nameuse{clist_map_inline:Nn}\@raw@classoptionslist{%
4409   \in@{=}{#1}%
4410   \ifin@else
4411     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4412   \fi}

```

Store the class/package options in a list. If there is an explicit main, it's placed as the last option. Then loop it to read the tex files, which can have a \BabelDefinitionFile. If there is no tex file, we attempt loading the ldf for the option name; if it fails, an error is raised. Note the option name is surrounded by //...//. Class and package options are separated with @@, because errors and info are dealt with in different ways. Consecutive identical languages count as one.

```

4413 \let\bbl@toload\@empty
4414 \let\bbl@toload@last\@empty
4415 \let\bbl@unkopt\@gobble %% <- Ugly
4416 \edef\bbl@tempc{%
4417   \bbl@tempf,@@,\bbl@language@opts
4418   \ifx\bbl@opt@main\@nnil\else,\bbl@opt@main\fi}
4419 \let\BabelLocalesTentative\bbl@tempc
4420 %
4421 \bbl@foreach\bbl@tempc{%
4422   \in@{@@}{#1}% <- Ugly
4423   \ifin@
4424     \def\bbl@unkopt##1{%
4425       \DeclareOption{##1}{\bbl@error{unknown-package-option}}{}}}%
4426   \else
4427     \def\CurrentOption{#1}%
4428     \bbl@xin@{/#1//}{\bbl@toload@last}% Collapse consecutive
4429     \ifin@else
4430       \lowercase{\InputIfFileExists{babel-#1.tex}}{ }{%
4431         \IfFileExists{#1.ldf}%
4432         {\edef\bbl@toload{%
4433           \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4434           \bbl@toload@last}%
4435           \edef\bbl@toload@last{0/0//\CurrentOption//und/#1}}%
4436         {\bbl@unkopt{#1}}}%
4437     \fi
4438   \fi}

```

We have to determine (1) if no language has been loaded (in which case we fallback to 'nil', with a special tag), and (2) the main language. With an explicit 'main' language, remove repeated elements. The number 1 flags it as the main language (relevant in *ini* locales), because with 0 becomes 10.

```

4439 \ifx\bbl@opt@main\@nnil
4440   \ifx\bbl@toload@last\@empty
4441     \def\bbl@toload@last{0/0//nil//und-x-nil/nil}
4442     \bbl@info{%

```

```

4443     You haven't specified a language as a class or package\\%
4444     option. I'll load 'nil'. Reported}
4445 \fi
4446 \else
4447 \let\bbl@tempc\@empty
4448 \bbl@foreach\bbl@toload{%
4449 \bbl@xin@{//\bbl@opt@main//}{#1}%
4450 \ifin@else
4451 \bbl@add@list\bbl@tempc{#1}%
4452 \fi}
4453 \let\bbl@toload\bbl@tempc
4454 \fi
4455 \edef\bbl@toload{\bbl@toload,1\bbl@toload@last}

```

Finally, load the ‘ini’ file or the pair ‘ini’/‘ldf’ file. Babel resorts to its own mechanism, not the default one based on \ProcessOptions (which is still present to make some internal clean-up). First, handle provide= and friends (with a recursive call if they are present), and then provide=* and friend. \count@ is used as flag: 0 if ‘ini’, 1 if ‘ldf’.

```

4456 \def\AfterBabelLanguage#1{%
4457 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}}
4458 \NewHook{babel/presets}
4459 \UseHook{babel/presets}
4460 %
4461 \let\bbl@tempb\@empty
4462 \def\bbl@tempc#1/#2/#3/#4/#5\@@{%
4463 \count@z@
4464 \ifnum#2=\@m % if no \BabelDefinitionFile
4465 \ifnum#1=z@ % not main. -- % if provide+=!, provide*=!
4466 \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0/#3/#4/#3\@@
4467 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4468 \fi
4469 \else % 10 = main -- % if provide+=!, provide*=!
4470 \ifodd\bbl@ldfflag\bbl@tempc 10/0/#3/#4/#3\@@
4471 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4472 \fi
4473 \fi
4474 \else
4475 \ifnum#1=z@ % not main
4476 \ifnum\bbl@iniflag>\@ne\else % if 0, provide
4477 \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4478 \fi
4479 \else % 10 = main
4480 \ifodd\bbl@iniflag\else % if provide+, provide*
4481 \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4482 \fi
4483 \fi
4484 \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4485 \fi}

```

Based on the value of \count@, do the actual loading. If ‘ldf’, we load the basic info from the ‘ini’ file before.

```

4486 \def\bbl@tempd#1#2#3#4#5{%
4487 \DeclareOption{#3}{%
4488 \ifcase\count@
4489 \bbl@exp{\bbl@add\bbl@tempb{%
4490 \global\let\bbl@afterload\@empty
4491 \bbl@nameuse\bbl@preini{#3}%
4492 \bbl@ldffinit
4493 \def\bbl@CurrentOption{#3}%
4494 \bbl@babelprovide[import=#4,\ifnum#1=z@\else\bbl@opt@provide,main\fi]{#3}%
4495 \bbl@afterldf
4496 \bbl@afterload}}%
4497 \else
4498 \bbl@add\bbl@tempb{%

```

```

4499 \global\let\bbl@afterload\@empty
4500 \def\CurrentOption{#3}%
4501 \let\localename\CurrentOption
4502 \let\language\localename
4503 \def\BabelIniTag{#4}%
4504 \@nameuse{bbl@preldf@#3}%
4505 \begingroup
4506 \bbl@id@assign
4507 \bbl@read@ini{\BabelIniTag}0%
4508 \endgroup
4509 \bbl@load@language{#5}%
4510 \bbl@afterload}%
4511 \fi}
4512 %
4513 \bbl@foreach\bbl@toload{\bbl@tempc#1\@@}
4514 \bbl@tempb
4515 \DeclareOption*{}
4516 \ProcessOptions
4517 %
4518 \bbl@exp{%
4519 \\\AtBeginDocument{\bbl@usehooks@lang{/}{\begindocument}{}}}%
4520 \def\AfterBabelLanguage{\bbl@error{late-after-babel}}%
4521 \AddToHook{begindocument/end}{%
4522 \bbl@info{Main language set to '\mainlocalename'\@gobble}}
4523 (/package)

```

5. Tagging

Preliminary and experimental. Only `\foreignlanguage` and `\selectlanguage`, for the moment. Requires the package option `babel-metadata=tagging`. Also changes for `\DocumentMetadata`. Option `babel-metadata=on` makes these changes without activating the changes for tagging.

```

4524 (*tagging)
4525 \ifx\bbl@chfmt\@undefined\else
4526 \EditInstance{heading}{chapter}{%
4527   prefix = ,
4528   number-format = \unskip\bbl@chfmt
4529 }
4530 \fi
4531 \ifx\bbl@partformat\@undefined\else
4532 \EditInstance{heading}{part}{%
4533   prefix = ,
4534   number-format = \unskip\bbl@partformat
4535 }
4536 \fi
4537 \ifpackagewith{babel}{babel-metadata=tagging}{\endinput}
4538 \let\bbl@ifmetadata\IfDocumentMetadataT
4539 \ExplSyntaxOn

```

`/WritingMode` attributes for bidi documents. Still only foreign and document. Select is pending.

```

4540 \ifnum\bbl@bidimode>\z@
4541 \tagpdfsetup{
4542   role/new-attribute = {babel-dir-rl}{/0/Layout/WritingMode/RLTb},
4543   role/new-attribute = {babel-dir-lr}{/0/Layout/WritingMode/LrTb} }
4544 \fi

```

Layout attributes (like `/WritingMode`) belong to structure elements, and cannot be used in the document's global catalog dictionary. (Note a redundant PDF resource is created.) The global `/Lang` is set by `\DocumentMetadata`, so no additional action is required here.

```

4545 \AddToHook{begindocument/end}{%
4546 \ifnum\bbl@bidimode>\z@
4547 \ifcase\bbl@thepardir\else
4548 \tag_struct_gput:nnn{\tag_get:n{struct_num}}{attribute}{/0/Layout/WritingMode/RLTb}%

```

```

4549 \fi
4550 \fi}

```

The code for `\foreignlanguage`. The attribute `/WritingMode` is set always (both LTR and RTL) in documents with some bidi text.

```

4551 \NewTaggingSocket{babel/foreign/begin}{0}
4552 \NewTaggingSocket{babel/foreign/end}{0}
4553 \NewTaggingSocketPlug{babel/foreign/begin}{default}{
4554 \tag_mc_end_push:
4555 \bbl@array@get\bbl@select@opts{phoneme}\bbl@tempa
4556 \ifx\bbl@tempa\relax\let\bbl@tempa\empty\fi
4557 \bbl@array@get\bbl@select@opts{actualtext}\bbl@tempb
4558 \ifx\bbl@tempb\relax\let\bbl@tempb\empty\fi
4559 \bbl@exp{%
4560 \tagstructbegin{tag=Span, lang=\BCPdata{tag},
4561 \ifnum\bbl@bidimode>\z@
4562 attribute=babel-dir-\ifcase\bbl@thetextdir lr\else rl\fi,
4563 \fi
4564 phoneme={\bbl@tempa}, actualtext={\bbl@tempb}}}%
4565 \tagmcbegin {} }
4566 \NewTaggingSocketPlug{babel/foreign/end}{default}{
4567 \tagmcend
4568 \tagstructend
4569 \tag_mc_begin_pop:n {}}
4570 \AssignTaggingSocketPlug{babel/foreign/begin}{default}
4571 \AssignTaggingSocketPlug{babel/foreign/end}{default}
4572 \ExplSyntaxOff

```

The code for `\selectlanguage`. For the moment, it only works in vertical mode. If the current language is the same as the default document language, it's set to nothing, i.e., the default. Note `\end{selectlanguage}` may get executed in horizontal mode. To be fixed.

```

4573 \AddBabelHook{babel-pdf tagging}{afterextras}{%
4574 \IfBabelSelectorTF{select}%
4575 {\ifvmode
4576 \bbl@ifsamestring{\BCPdata{tag}}{\BCPdata{main.tag}}}%
4577 {\tagpdfsetup{text/lang=}}
4578 {\tagpdfsetup{text/lang=\BCPdata{tag}}}%
4579 \else
4580 \bbl@once{select-tag-vmode}{%
4581 \bbl@warning{%
4582 PDF language tagging with \string\selectlanguage\%
4583 requires vertical mode. Reported}}%
4584 \fi}%
4585 {}}
4586 (/tagging)

```

6. Error messages

They are loaded when `\bll@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.

```

4587 (*errors)
4588 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4589 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4590 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4591 \catcode`\@=11 \catcode`\^=7
4592 %
4593 \ifx\MessageBreak\undefined
4594 \gdef\bbl@error@i#1#2{%
4595 \begingroup
4596 \newlinechar=`^^J

```

```

4597     \def\{^J(babel) }%
4598     \errhelp{#2}\errmessage{\#1}%
4599     \endgroup}
4600 \else
4601     \gdef\bbl@error@i#1#2{%
4602         \begingroup
4603         \def\{MessageBreak}%
4604         \PackageError{babel}{#1}{#2}%
4605         \endgroup}
4606 \fi
4607 \def\bbl@errmessage#1#2#3{%
4608     \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4609         \bbl@error@i{#2}{#3}}
4610 % Implicit #2#3#4:
4611 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4612 %
4613 \bbl@errmessage{not-yet-available}
4614     {Not yet available}%
4615     {Find an armchair, sit down and wait}
4616 \bbl@errmessage{bad-package-option}%
4617     {Bad option '#1=#2'. Either you have misspelled the\\%
4618     key or there is a previous setting of '#1'. Valid\\%
4619     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4620     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4621     {See the manual for further details.}
4622 \bbl@errmessage{base-on-the-fly}
4623     {For a language to be defined on the fly 'base'\\%
4624     is not enough, and the whole package must be\\%
4625     loaded. Either delete the 'base' option or\\%
4626     request the languages explicitly}%
4627     {See the manual for further details.}
4628 \bbl@errmessage{undefined-language}
4629     {You haven't defined the language '#1' yet.\\%
4630     Perhaps you misspelled it or your installation\\%
4631     is not complete}%
4632     {Your command will be ignored, type <return> to proceed}
4633 \bbl@errmessage{invalid-ini-name}
4634     {'#1' not valid with the 'ini' mechanism.\\%
4635     I think you want '#2' instead. You may continue,\\%
4636     but you should fix the name. See the babel manual\\%
4637     for the available locales with 'provide'}%
4638     {See the manual for further details.}
4639 \bbl@errmessage{shorthand-is-off}
4640     {I can't declare a shorthand turned off (\string#2)}
4641     {Sorry, but you can't use shorthands which have been\\%
4642     turned off in the package options}
4643 \bbl@errmessage{not-a-shorthand}
4644     {The character '\string #1' should be made a shorthand character;\\%
4645     add the command \string\usesshorthands\string{#1\string} to
4646     the preamble.\\%
4647     I will ignore your instruction}%
4648     {You may proceed, but expect unexpected results}
4649 \bbl@errmessage{not-a-shorthand-b}
4650     {I can't switch '\string#2' on or off--not a shorthand\\%
4651     This character is not a shorthand. Maybe you made\\%
4652     a typing mistake?}%
4653     {I will ignore your instruction.}
4654 \bbl@errmessage{unknown-attribute}
4655     {The attribute #2 is unknown for language #1.}%
4656     {Your command will be ignored, type <return> to proceed}
4657 \bbl@errmessage{missing-group}
4658     {Missing group for string \string#1}%
4659     {You must assign strings to some category, typically\\%

```

```

4660     captions or extras, but you set none}
4661 \bbl@errmessage{only-lua-xe}
4662 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4663 {Consider switching to these engines.}
4664 \bbl@errmessage{only-lua}
4665 {This macro is available only in LuaLaTeX}%
4666 {Consider switching to that engine.}
4667 \bbl@errmessage{unknown-provide-key}
4668 {Unknown key '#1' in \string\babelprovide}%
4669 {See the manual for valid keys}%
4670 \bbl@errmessage{unknown-mapfont}
4671 {Option '\bbl@KVP@mapfont' unknown for\\%
4672   mapfont. Use 'direction'}%
4673 {See the manual for details.}
4674 \bbl@errmessage{no-ini-file}
4675 {There is no ini file for the requested language\\%
4676   (#1: \language). Perhaps you misspelled it or your\\%
4677   installation is not complete}%
4678 {Fix the name or reinstall babel.}
4679 \bbl@errmessage{digits-is-reserved}
4680 {The counter name 'digits' is reserved for mapping\\%
4681   decimal digits}%
4682 {Use another name.}
4683 \bbl@errmessage{limit-two-digits}
4684 {Currently two-digit years are restricted to the\\5
4685   range 0-9999}%
4686 {There is little you can do. Sorry.}
4687 \bbl@errmessage{counter-too-large}
4688 {Counter too large (#1)}%
4689 {Currently this is the limit.}
4690 \bbl@errmessage{no-ini-info}
4691 {I've found no info for the current locale.\\%
4692   The corresponding ini file has not been loaded\\%
4693   Perhaps it doesn't exist}%
4694 {See the manual for details.}
4695 \bbl@errmessage{unknown-ini-field}
4696 {Unknown field '#1' in \string\BCPdata.\\%
4697   Perhaps you misspelled it}%
4698 {See the manual for details.}
4699 \bbl@errmessage{unknown-locale-key}
4700 {Unknown key for locale '#2':\\%
4701   #3\\%
4702   \string#1 will be set to \string\relax}%
4703 {Perhaps you misspelled it.}%
4704 \bbl@errmessage{adjust-only-vertical}
4705 {Currently, #1 related features can be adjusted only\\%
4706   in the main vertical list}%
4707 {Maybe things change in the future, but this is what it is.}
4708 \bbl@errmessage{layout-only-vertical}
4709 {Currently, layout related features can be adjusted only\\%
4710   in vertical mode}%
4711 {Maybe things change in the future, but this is what it is.}
4712 \bbl@errmessage{bidi-only-lua}
4713 {The bidi method 'basic' is available only in\\%
4714   luatex. I'll continue with 'bidi=default', so\\%
4715   expect wrong results.\\%
4716   Suggested actions:\\%
4717   * If possible, switch to luatex, as xetex is not\\%
4718   recommend anymore.\\%
4719   * If you can't, try 'bidi=bidi' with xetex.\\%
4720   * With pdftex, only 'bidi=default' is available.}%
4721 {See the manual for further details.}
4722 \bbl@errmessage{multiple-bidi}

```

```

4723 {Multiple bidi settings inside a group\\%
4724 I'll insert a new group, but expect wrong results.\\%
4725 Suggested action:\\%
4726 * Add a new group where appropriate.}
4727 {See the manual for further details.}
4728 \bbl@errmessage{unknown-package-option}
4729 {Unknown option '\CurrentOption'.\\%
4730 Suggested actions:\\%
4731 * Make sure you haven't misspelled it\\%
4732 * Check in the babel manual that it's supported\\%
4733 * If supported and it's a language, you may\\%
4734 \space\space need in some distributions a separate\\%
4735 \space\space installation\\%
4736 * If installed, check there isn't an old\\%
4737 \space\space version of the required files in your system\\%
4738 * If it's an unsupported language, create it with\\%
4739 \space\space \string\babelprovide\space (see the manual)}
4740 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4741 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4742 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4743 \bbl@errmessage{config-not-found}
4744 {Local config file '\bbl@opt@config.cfg' not found.\\%
4745 Suggested actions:\\%
4746 * Make sure you haven't misspelled it in config=\\%
4747 * Check it exists and it's in the correct path}%
4748 {Perhaps you misspelled it.}
4749 \bbl@errmessage{late-after-babel}
4750 {Too late for \string\AfterBabelLanguage}%
4751 {Languages have been loaded, so I can do nothing}
4752 \bbl@errmessage{double-hyphens-class}
4753 {Double hyphens aren't allowed in \string\babelcharclass\\%
4754 because it's potentially ambiguous}%
4755 {See the manual for further info}
4756 \bbl@errmessage{unknown-interchar}
4757 {'#1' for '\language' cannot be enabled.\\%
4758 Maybe there is a typo}%
4759 {See the manual for further details.}
4760 \bbl@errmessage{unknown-interchar-b}
4761 {'#1' for '\language' cannot be disabled.\\%
4762 Maybe there is a typo}%
4763 {See the manual for further details.}
4764 \bbl@errmessage{charproperty-only-vertical}
4765 {\string\babelcharproperty\space can be used only in\\%
4766 vertical mode (preamble or between paragraphs)}%
4767 {See the manual for further info}
4768 \bbl@errmessage{unknown-char-property}
4769 {No property named '#2'. Allowed values are\\%
4770 direction (bc), mirror (bmg), and linebreak (lb)}%
4771 {See the manual for further info}
4772 \bbl@errmessage{bad-transform-option}
4773 {Bad option '#1' in a transform.\\%
4774 I'll ignore it but expect more errors}%
4775 {See the manual for further info.}
4776 \bbl@errmessage{font-conflict-transforms}
4777 {Transforms cannot be re-assigned to different\\%
4778 fonts. The conflict is in '\bbl@kv@label'.\\%
4779 Apply the same fonts or use a different label}%
4780 {See the manual for further details.}
4781 \bbl@errmessage{transform-not-available}
4782 {'#1' for '\language' cannot be enabled.\\%
4783 Maybe there is a typo or it's a font-dependent transform}%
4784 {See the manual for further details.}
4785 \bbl@errmessage{transform-not-available-b}

```

```

4786 {'#1' for '\language' cannot be disabled.\\%
4787 Maybe there is a typo or it's a font-dependent transform}%
4788 {See the manual for further details.}
4789 \bbl@errmessage{year-out-range}
4790 {Year out of range.\\%
4791 The allowed range is #1}%
4792 {See the manual for further details.}
4793 \bbl@errmessage{only-pdfTeX-lang}
4794 {The '#1' ldf style doesn't work with #2,\\%
4795 but you can use the ini locale instead.\\%
4796 Try adding 'provide=*' to the option list. You may\\%
4797 also want to set 'bidi=' to some value}%
4798 {See the manual for further details.}
4799 \bbl@errmessage{hyphenmins-args}
4800 {\string\babelhyphenmins\ accepts either the optional\\%
4801 argument or the star, but not both at the same time}%
4802 {See the manual for further details.}
4803 \bbl@errmessage{no-locale-for-meta}
4804 {There isn't currently a locale for the 'lang' requested\\%
4805 in the PDF metadata ('#1'). To fix it, you can\\%
4806 set explicitly a similar language (using the same\\%
4807 script) with the key main= when loading babel. If you\\%
4808 continue, I'll fallback to the 'nil' language, with\\%
4809 tag 'und' and script 'Latn', but expect a bad font\\%
4810 rendering with other scripts. You may also need set\\%
4811 explicitly captions and date, too}%
4812 {See the manual for further details.}
4813 (/errors)
4814 (*patterns)

```

7. 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.

```

4815 <@Make sure ProvidesFile is defined@>
4816 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4817 \xdef\bbl@format{\jobname}
4818 \def\bbl@version{<@version@>}
4819 \def\bbl@date{<@date@>}
4820 \ifx\AtBeginDocument\undefined
4821 \def\@empty{}
4822 \fi
4823 <@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.

```

4824 \def\process@line#1#2 #3 #4 {%
4825   \ifx=#1%
4826     \process@synonym{#2}%
4827   \else
4828     \process@language{#1#2}{#3}{#4}%
4829   \fi
4830   \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.

```

4831 \toks@{}
4832 \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.

```

4833 \def\process@synonym#1{%
4834   \ifnum\last@language=\m@ne
4835     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4836   \else
4837     \expandafter\chardef\csname l@#1\endcsname\last@language
4838     \wlog{\string\l@#1=\string\language\the\last@language}%
4839     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4840       \csname\language\hyphenmins\endcsname
4841     \let\bbl@elt\relax
4842     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4843   \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` or `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

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

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

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

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

```

4844 \def\process@language#1#2#3{%
4845   \expandafter\addlanguage\csname l@#1\endcsname
4846   \expandafter\language\csname l@#1\endcsname
4847   \edef\language{#1}%
4848   \bbl@hook@everylanguage{#1}%
4849   % > luatex
4850   \bbl@get@enc#1: :@@@
4851   \begingroup
4852     \lefthyphenmin\m@ne
4853     \bbl@hook@loadpatterns{#2}%
4854     % > luatex
4855     \ifnum\lefthyphenmin=\m@ne
4856     \else
4857       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4858         \the\lefthyphenmin\the\righthyphenmin}%
4859     \fi
4860   \endgroup
4861   \def\bbl@tempa{#3}%

```

```

4862 \ifx\bbl@tempa\@empty\else
4863   \bbl@hook@loadexceptions{#3}%
4864   % > latex
4865 \fi
4866 \let\bbl@elt\relax
4867 \edef\bbl@languages{%
4868   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4869 \ifnum\the\language=\z@
4870   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4871     \set@hyphenmins\tw@\thr@@\relax
4872   \else
4873     \expandafter\expandafter\expandafter\set@hyphenmins
4874     \csname #1hyphenmins\endcsname
4875   \fi
4876   \the\toks@
4877   \toks@{}%
4878 \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.

```

4879 \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 latex, format-specific configuration files are taken into account. \bbl@hook@loadkernel currently loads nothing, but define some basic macros instead.

```

4880 \def\bbl@hook@everylanguage#1{}
4881 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4882 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4883 \def\bbl@hook@loadkernel#1{%
4884   \def\addlanguage{\csname newlanguage\endcsname}%
4885   \def\adddialect##1##2{%
4886     \global\chardef##1##2\relax
4887     \wlog{\string##1 = a dialect from \string\language##2}}%
4888   \def\iflanguage##1{%
4889     \expandafter\ifx\csname l@##1\endcsname\relax
4890       \@nolanerr{##1}%
4891     \else
4892       \ifnum\csname l@##1\endcsname=\language
4893         \expandafter\expandafter\expandafter\@firstoftwo
4894       \else
4895         \expandafter\expandafter\expandafter\@secondoftwo
4896       \fi
4897     \fi}%
4898   \def\providehyphenmins##1##2{%
4899     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4900       \@namedef{##1hyphenmins}{##2}%
4901     \fi}%
4902   \def\set@hyphenmins##1##2{%
4903     \lefthyphenmin##1\relax
4904     \righthyphenmin##2\relax}%
4905   \def\selectlanguage{%
4906     \errhelp{Selecting a language requires a package supporting it}%
4907     \errmessage{No multilingual package has been loaded}}%
4908   \let\foreignlanguage\selectlanguage
4909   \let\otherlanguage\selectlanguage
4910   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4911   \def\bbl@usehooks##1##2{%
4912     \def\setlocale{%
4913       \errhelp{Find an armchair, sit down and wait}%
4914       \errmessage{(babel) Not yet available}}%
4915     \let\uselocale\setlocale
4916     \let\locale\setlocale

```

```

4917 \let\selectlocale\setlocale
4918 \let\localename\setlocale
4919 \let\textlocale\setlocale
4920 \let\textlanguage\setlocale
4921 \let\languagetext\setlocale}
4922 \beginingroup
4923 \def\AddBabelHook#1#2{%
4924   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4925     \def\next{\toks1}%
4926   \else
4927     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4928   \fi
4929   \next}
4930 \ifx\directlua\@undefined
4931   \ifx\XeTeXinputencoding\@undefined\else
4932     \input xebabel.def
4933   \fi
4934 \else
4935   \input luababel.def
4936 \fi
4937 \openin1 = babel-\bbl@format.cfg
4938 \ifeof1
4939 \else
4940   \input babel-\bbl@format.cfg\relax
4941 \fi
4942 \closein1
4943 \endgroup
4944 \bbl@hook@loadkernel{switch.def} % Dummy argument

```

readconfigfile The configuration file can now be opened for reading.

```

4945 \openin1 = language.dat

```

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

```

4946 \def\language{english}%
4947 \ifeof1
4948   \message{I couldn't find the file language.dat,\space
4949     I will try the file hyphen.tex}
4950   \input hyphen.tex\relax
4951   \chardef\l@english\z@
4952 \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`.

```

4953 \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.

```

4954 \loop
4955   \endlinechar\m@ne
4956   \read1 to \bbl@line
4957   \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.

```

4958   \if T\ifeof1\fi T\relax
4959   \ifx\bbl@line\@empty\else
4960     \edef\bbl@line{\bbl@line\space\space\space}%
4961     \expandafter\process@line\bbl@line\relax

```

```

4962     \fi
4963 \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.

```

4964 \begingroup
4965   \def\bbl@elt#1#2#3#4{%
4966     \global\language=#2\relax
4967     \gdef\languagename{#1}%
4968     \def\bbl@elt##1##2##3##4{}}%
4969   \bbl@languages
4970 \endgroup
4971 \fi
4972 \closein1

```

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

```

4973 \if/\the\toks@\else
4974   \errhelp{language.dat loads no language, only synonyms}
4975   \errmessage{Orphan language synonym}
4976 \fi

```

Also remove some macros from memory and raise an error if `\toks@` is not empty.

```

4977 \let\bbl@line\@undefined
4978 \let\process@line\@undefined
4979 \let\process@synonym\@undefined
4980 \let\process@language\@undefined
4981 \let\bbl@get@enc\@undefined
4982 \let\bbl@hyph@enc\@undefined
4983 \let\bbl@tempa\@undefined
4984 \let\bbl@hook@loadkernel\@undefined
4985 \let\bbl@hook@everylanguage\@undefined
4986 \let\bbl@hook@loadpatterns\@undefined
4987 \let\bbl@hook@loadexceptions\@undefined
4988 (/patterns)

```

Here the code for `iniTeX` ends.

8. **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`).

```

4989 ((*More package options)) ▶
4990 \chardef\bbl@bidimode\z@
4991 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4992 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4993 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4994 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4995 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4996 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4997 \DeclareOption{bidi=unibidi}{\chardef\bbl@bidimode=301 }
4998 (/More package options)

```

\babbelfont 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`.

```

4999 ((*Font selection)) ▶
5000 \bbl@trace{Font handling with fontspec}
5001 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5002 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckcheckstdfonts}
5003 \DisableBabelHook{babel-fontspec}

```

```

5004 \@onlypreamble\babelfont
5005 \ifx\NewDocumentCommand\undefined\else % Not plain
5006 \NewDocumentCommand\babelfont{0}{m0}{m0}}{%
5007 \bbl@bblfont@o[#1]{#2}[#3,#5]{#4}}
5008 \fi
5009 \newcommand\bbl@bblfont@o[2][]{% 1=langs/scripts 2=fam
5010 \ifx\fontspec\undefined
5011 \usepackage{fontspec}%
5012 \fi
5013 \EnableBabelHook{babel-fontspec}%
5014 \edef\bbl@tempa{#1}%
5015 \def\bbl@tempb{#2}% Used by \bbl@bblfont
5016 \bbl@bblfont}
5017 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
5018 \bbl@ifunset{\bbl@tempb family}%
5019 {\bbl@providefam{\bbl@tempb}}%
5020 {}%
5021 % For the default font, just in case:
5022 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
5023 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
5024 {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
5025 \bbl@exp{%
5026 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
5027 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
5028 \<\bbl@tempb default>\<\bbl@tempb family>}}%
5029 {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
5030 \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:

```

5031 \def\bbl@providefam#1{%
5032 \bbl@exp{%
5033 \\\newcommand<#1default>{}% Just define it
5034 \\\bbl@add@list\\bbl@font@fams{#1}%
5035 \\\NewHook{#1family}%
5036 \\\DeclareRobustCommand<#1family>{%
5037 \\\not@math@alphabet<#1family>\relax
5038 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
5039 \\\fontfamily<#1default>%
5040 \\\UseHook{#1family}%
5041 \\\selectfont}%
5042 \\\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.

```

5043 \def\bbl@nostdfont#1{%
5044 \bbl@once{nostdfam-\f@family}%
5045 {\bbl@infowarn{The current font is not a babel standard family:\%
5046 #1%
5047 \fontname\font\\%
5048 There is nothing intrinsically wrong, and you can\\%,
5049 ignore this message altogether if you do not need\\%
5050 this font. If they are used in the document, be aware\\%
5051 'babel' will not set Script and Language for it, so\\%
5052 you may consider defining a new family with \string\babelfont.\\%
5053 See the manual for further details about \string\babelfont.
5054 Reported}}%
5055 }%
5056 \gdef\bbl@switchfont{%
5057 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
5058 \bbl@exp{% e.g., Arabic -> arabic
5059 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
5060 \bbl@foreach\bbl@font@fams{%
5061 \bbl@ifunset{\bbl@##1dflt@\languagename}% (1) language?
5062 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?

```

```

5063      {\bbl@ifunset{\bbl@##ldflt@}%          2=F - (3) from generic?
5064      {}}%                                   123=F - nothing!
5065      {\bbl@exp{%                            3=T - from generic
5066      \global\let\<bbl@##ldflt@\language\>%
5067      \<bbl@##ldflt@>}}}%
5068      {\bbl@exp{%                            2=T - from script
5069      \global\let\<bbl@##ldflt@\language\>%
5070      \<bbl@##ldflt@*\bbl@tempa>}}}%
5071      {}}%                                   1=T - language, already defined
5072 \def\bbl@tempa{\bbl@nostdfont{}}%
5073 \bbl@foreach\bbl@font@fams{%    don't gather with prev for
5074 \bbl@ifunset{\bbl@##ldflt@\language\>%
5075 {\bbl@cs{famrst@##1}%
5076 \global\bbl@csarg\let{famrst@##1}\relax}%
5077 {\bbl@exp{% order is relevant.
5078 \\\bbl@add\\\originalTeX{%
5079 \\\bbl@font@rst{\bbl@cl{##ldflt}}}%
5080 \<##lddefault>\<##1family>{##1}}}%
5081 \\\bbl@font@set\<bbl@##ldflt@\language\>% the main part!
5082 \<##lddefault>\<##1family>}}}%
5083 \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`.

```

5084 \ifx\f@family\undefined\else    % if latex
5085 \ifcase\bbl@engine                % if pdftex
5086 \let\bbl@ckeckstdfonts\relax
5087 \else
5088 \def\bbl@ckeckstdfonts{%
5089 \begingroup
5090 \global\let\bbl@ckeckstdfonts\relax
5091 \let\bbl@tempa\@empty
5092 \bbl@foreach\bbl@font@fams{%
5093 \bbl@ifunset{\bbl@##ldflt@}%
5094 {\@nameuse{##1family}}%
5095 \bbl@csarg\gdef{WFF@f@family}}}% Flag
5096 \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\}%
5097 \space\space\fontname\font\\}%
5098 \bbl@csarg\xdef{##ldflt@}{f@family}%
5099 \expandafter\xdef\csname ##lddefault\endcsname{f@family}}}%
5100 {}}%
5101 \ifx\bbl@tempa\@empty\else
5102 \bbl@infowarn{The following font families will use the default\\%
5103 settings for all or some languages:\\%
5104 \bbl@tempa
5105 There is nothing intrinsically wrong with it, but\\%
5106 'babel' will no set Script and Language, which could\\%
5107 be relevant in some languages. If your document uses\\%
5108 these families, consider redefining them with \string\babelfont.\\%
5109 Reported}%
5110 \fi
5111 \endgroup}
5112 \fi
5113 \fi

```

Now the macros defining the font with `fontspec`.

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

For historical reasons, $\text{\LaTeX}$ can select two different series (bx and b), for what is conceptually a single one. This can lead to problems when a single family requires several fonts, depending on the language, mainly because ‘substitutions’ with some combinations are not done consistently – sometimes bx/sc is the correct font, but sometimes points to b/n, even if b/sc exists. So, some

substitutions are redefined (in a somewhat hackish way, by inspecting if the variant declaration contains >ssub*).

```

5114 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
5115   \bbl@xin@{<>}{#1}%
5116   \ifin@
5117     \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
5118   \fi
5119   \bbl@exp{%
5120     \def\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
5121     \bbl@ifsamestring{#2}{\f@family}%
5122     {\#3%
5123       \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}{}%
5124       \let\bbl@tempa\relax}%
5125     {}%

```

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).

```

5126 \def\bbl@fontspec@lsys{%
5127   \ifx\newfontscript\@undefined\else\bbl@exp{%
5128     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
5129     {\bbl@cl{sname}}{\bbl@cl{sotf}}}%
5130   \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
5131     {\bbl@cl{lname}}{\bbl@cl{lotf}}}%
5132   \fi}
5133 \ifx\AddToHook\@undefined
5134   \let\bbl@fontspec@lsys\relax
5135 \else
5136   \AddToHook{selectfont}{\bbl@fontspec@lsys}
5137 \fi
5138 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
5139   \let\bbl@tempe\bbl@mapselect
5140   \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
5141   \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}}%
5142   \let\bbl@mapselect\relax
5143   \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
5144   \let#4\@empty % Make sure \renewfontfamily is valid
5145   \bbl@set@renderer
5146   \bbl@fontspec@lsys
5147   \bbl@exp{%
5148     \let\bbl@temp@pfam<\bbl@stripslash#4\space> e.g., '\rmfamily '
5149     \renewfontfamily\#4%
5150     [\bbl@cl{lsys},% xetex removes unknown features :-(
5151       \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
5152       #2]}{#3}% i.e., \bbl@exp{..}{#3}
5153   \bbl@unset@renderer
5154   \begingroup
5155     #4%
5156     \xdef#1{\f@family}% e.g., \bbl@rmdflt@lang{FreeSerif(0)}
5157   \endgroup
5158   \bbl@xin@{\detokenize{>ssub*}}%
5159   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
5160   \ifin@
5161     \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
5162   \fi
5163   \bbl@xin@{\detokenize{>ssub*}}%
5164   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
5165   \ifin@
5166     \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
5167   \fi
5168   \let#4\bbl@temp@fam

```

```

5169 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
5170 \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.

```

5171 \def\bbl@font@rst#1#2#3#4{%
5172 \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

5173 \def\bbl@font@fams{rm,sf,tt}
5174 ({/Font selection})

```

9. Hooks for XeTeX and LuaTeX

9.1. XeTeX

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

Now, the code.

```

5175 (*xetex)
5176 \def\BabelStringsDefault{unicode}
5177 \let\xebbl@stop\relax
5178 \AddBabelHook{xetex}{encodedcommands}{%
5179 \def\bbl@tempa{#1}%
5180 \ifx\bbl@tempa@empty
5181 \XeTeXinputencoding"bytes"%
5182 \else
5183 \XeTeXinputencoding"#1"%
5184 \fi
5185 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
5186 \AddBabelHook{xetex}{stopcommands}{%
5187 \xebbl@stop
5188 \let\xebbl@stop\relax}
5189 \def\bbl@input@classes{% Used in CJK intraspaces
5190 \input{load-unicode-xetex-classes.tex}%
5191 \let\bbl@input@classes\relax}
5192 \def\bbl@intraspace#1 #2 #3\@@{%
5193 \bbl@csarg\gdef{xeisp@\languagename}%
5194 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
5195 \def\bbl@intrapenalty#1\@@{%
5196 \bbl@csarg\gdef{xeipn@\languagename}%
5197 {\XeTeXlinebreakpenalty #1\relax}}
5198 \def\bbl@provide@intraspace{%
5199 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
5200 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
5201 \ifin@
5202 \bbl@ifunset{bbl@intsp@\languagename}{}%
5203 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5204 \ifx\bbl@KVP@intraspace\@nnil
5205 \bbl@exp{%
5206 \\\bbl@intraspace\bbl@cl{intsp}\\\@}%
5207 \fi
5208 \ifx\bbl@KVP@intrapenalty\@nnil
5209 \bbl@intrapenalty0\@@
5210 \fi
5211 \fi
5212 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
5213 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
5214 \fi
5215 \ifx\bbl@KVP@intrapenalty\@nnil\else
5216 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@

```

```

5217 \fi
5218 \bbl@exp{%
5219   \\bbl@add<extras\language>{%
5220     \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
5221     \<bbl@xeisp@<language>%
5222     \<bbl@xeipn@<language>%
5223     \\bbl@tglobal\<extras\language>%
5224     \\bbl@add<noextras\language>{%
5225       \XeTeXlinebreaklocale ""}%
5226     \\bbl@tglobal\<noextras\language>%
5227   \ifx\bbl@ispace\undefined
5228     \gdef\bbl@ispace{\bbl@cl{xeisp}}%
5229   \ifx\AtBeginDocument\@notprerr
5230     \expandafter\@secondoftwo % to execute right now
5231   \fi
5232   \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
5233 \fi}%
5234 \fi}
5235 \ifx\DisableBabelHook\undefined\endinput\fi
5236 \let\bbl@set@renderer\relax
5237 \let\bbl@unset@renderer\relax
5238 <@Font selection@>
5239 \def\bbl@provide@extra#1{}

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

5240 \def\bbl@xeno\hyphenatd{%
5241   \bbl@ifset{\bbl@prehc@<language>%
5242     {\ifnum\hyphenchar\font=\default\hyphenchar
5243       \iffontchar\font\bbl@cl{prehc}\relax
5244       \hyphenchar\font\bbl@cl{prehc}\relax
5245       \else\iffontchar\font"200B
5246         \hyphenchar\font"200B
5247       \else
5248         \bbl@warning
5249           {Neither 0 nor ZERO WIDTH SPACE are available\\%
5250             in the current font, and therefore the hyphen\\%
5251             will be printed. Try changing the fontspec's\\%
5252             'HyphenChar' to another value, but be aware\\%
5253             this setting is not safe (see the manual).\\%
5254             Reported}%
5255         \hyphenchar\font\default\hyphenchar
5256       \fi\fi
5257     \fi}%
5258   {\hyphenchar\font\default\hyphenchar}}

```

9.2. 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.

```

5259 \ifnum\Xe@alloc@intercharclass<\thr@@
5260   \Xe@alloc@intercharclass\thr@@
5261 \fi
5262 \chardef\bbl@xe@class@default=\z@
5263 \chardef\bbl@xe@class@cjkideogram=\@ne
5264 \chardef\bbl@xe@class@cjkleftpunctuation=\tw@
5265 \chardef\bbl@xe@class@cjkrightpunctuation=\thr@@
5266 \chardef\bbl@xe@class@boundary=4095
5267 \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.

```

5268 \AddBabelHook{babel-interchar}{beforeextras}{%
5269   \nameuse{bbl@xechars@\languagename}}
5270 \DisableBabelHook{babel-interchar}
5271 \protected\def\bbl@charclass#1{%
5272   \ifnum\count@<\z@
5273     \count@-\count@
5274     \loop
5275       \bbl@exp{%
5276         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5277         \XeTeXcharclass\count@ \bbl@tempc
5278         \ifnum\count@<`#1\relax
5279         \advance\count@ \@ne
5280       \repeat
5281   \else
5282     \babel@savevariable{\XeTeXcharclass`#1}%
5283     \XeTeXcharclass`#1 \bbl@tempc
5284   \fi
5285   \count@`#1\relax}

```

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

```

5286 \newcommand\bbl@ifinterchar[1]{%
5287   \let\bbl@tempa\@gobble           % Assume to ignore
5288   \edef\bbl@tempb{\zap@space#1 \@empty}%
5289   \ifx\bbl@KVP@interchar\@nnil\else
5290     \bbl@replace\bbl@KVP@interchar{ }{,}%
5291     \bbl@foreach\bbl@tempb{%
5292       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5293       \ifin@
5294       \let\bbl@tempa\@firstofone
5295     \fi}%
5296   \fi
5297   \bbl@tempa}
5298 \newcommand\IfBabelIntercharT[2]{%
5299   \bbl@carg\bbl@add{\bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5300 \newcommand\babelcharclass[3]{%
5301   \EnableBabelHook{babel-interchar}%
5302   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5303   \def\bbl@tempb##1{%
5304     \ifx##1\@empty\else
5305       \ifx##1-%
5306         \bbl@upto
5307       \else
5308         \bbl@charclass{%
5309           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5310         \fi
5311       \expandafter\bbl@tempb
5312     \fi}%
5313   \bbl@ifunset{\bbl@xechars@#1}%
5314   {\toks@{%
5315     \babel@savevariable\XeTeXinterchartokenstate
5316     \XeTeXinterchartokenstate\@ne
5317   }}%
5318   {\toks@\expandafter\expandafter\expandafter{%
5319     \csname bbl@xechars@#1\endcsname}}}%
5320   \bbl@csarg\edef{xechars@#1}{%
5321     \the\toks@
5322     \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5323     \bbl@tempb#3\@empty}}

```

```

5324 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
5325 \protected\def\bbl@upto{%
5326   \ifnum\count@>\z@
5327     \advance\count@\@ne
5328     \count@-\count@
5329   \else\ifnum\count@=\z@
5330     \bbl@charclass{-}%
5331   \else
5332     \bbl@error{double-hyphens-class}{}{}{}%
5333   \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>`.

```

5334 \def\bbl@ignoreinterchar{%
5335   \ifnum\language=\l@nohyphenation
5336     \expandafter\@gobble
5337   \else
5338     \expandafter\@firstofone
5339   \fi}
5340 \newcommand\babelinterchar[5][]{%
5341   \let\bbl@kv@label\@empty
5342   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5343   \@namedef{\zap@space \bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5344   {\bbl@ignoreinterchar{#5}}%
5345   \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5346   \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5347     \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5348       \XeTeXinterchartoks
5349         \@nameuse{\bbl@xeclass@\bbl@tempa @#2}
5350         \bbl@ifunset{\bbl@xeclass@\bbl@tempa @#2}{#2} %
5351         \@nameuse{\bbl@xeclass@\bbl@tempb @#2}
5352         \bbl@ifunset{\bbl@xeclass@\bbl@tempb @#2}{#2} %
5353       = \expandafter{%
5354         \csname \bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5355         \csname \zap@space \bbl@xeinter@\bbl@kv@label
5356           @#3@#4@#2 \@empty\endcsname}}}}
5357 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5358   \bbl@ifunset{\bbl@ic@#1\@languagename}%
5359   {\bbl@error{unknown-interchar}{#1}{}}%
5360   {\bbl@csarg\let{ic@#1\@languagename}\@firstofone}}
5361 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5362   \bbl@ifunset{\bbl@ic@#1\@languagename}%
5363   {\bbl@error{unknown-interchar-b}{#1}{}}%
5364   {\bbl@csarg\let{ic@#1\@languagename}\@gobble}}
5365 \xetex

```

9.3. Layout

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

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

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

```

5366 (*xetex|texxet)
5367 \providecommand\bbl@provide@intraspace{}
5368 \bbl@trace{Redefinitions for bidi layout}

```

Finish here if there in no layout.

```

5369 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5370   \ifx\bbl@opt@layout\@undefined\else % Plain

```

```

5371 \IfBabelLayout{nopars}
5372 {}
5373 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5374 \fi
5375 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5376 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5377 \ifnum\bbl@bidimode>\z@
5378 \IfBabelLayout{pars}
5379 {\def\@hangfrom#1{%
5380 \setbox\@tempboxa\hbox{#{#1}}%
5381 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5382 \noindent\box\@tempboxa}
5383 \def\raggedright{%
5384 \let\\\@centercr
5385 \bbl@startskip\z@skip
5386 \@rightskip\@flushglue
5387 \bbl@endskip\@rightskip
5388 \parindent\z@
5389 \parfillskip\bbl@startskip}
5390 \def\raggedleft{%
5391 \let\\\@centercr
5392 \bbl@startskip\@flushglue
5393 \bbl@endskip\z@skip
5394 \parindent\z@
5395 \parfillskip\bbl@endskip}}
5396 {}
5397 \fi
5398 \IfBabelLayout{lists}
5399 {\bbl@sreplace\list
5400 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5401 \def\bbl@listleftmargin{%
5402 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5403 \ifcase\bbl@engine
5404 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5405 \def\p@enumiii{\p@enumii}\theenumii{}\fi
5406 \fi
5407 \bbl@sreplace\@verbatim
5408 {\leftskip\@totalleftmargin}%
5409 {\bbl@startskip\textwidth
5410 \advance\bbl@startskip-\linewidth}%
5411 \bbl@sreplace\@verbatim
5412 {\rightskip\z@skip}%
5413 {\bbl@endskip\z@skip}}%
5414 {}
5415 \IfBabelLayout{contents}
5416 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5417 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5418 {}
5419 \IfBabelLayout{columns}
5420 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outpuhbox}%
5421 \def\bbl@outpuhbox#1{%
5422 \hb@xt@\textwidth{%
5423 \hskip\columnwidth
5424 \hfil
5425 {\normalcolor\vrule \@width\columnseprule}%
5426 \hfil
5427 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5428 \hskip-\textwidth
5429 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5430 \hskip\columnsep
5431 \hskip\columnwidth}}}%
5432 {}

```

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.

```

5433 \IfBabelLayout{counters*}%
5434 {\bbl@add\bbl@opt@layout{.counters.}%
5435 \AddToHook{shipout/before}{%
5436 \let\bbl@tempa\babelsublr
5437 \let\babelsublr\@firstofone
5438 \let\bbl@save@thepage\thepage
5439 \protected@edef\thepage{\thepage}%
5440 \let\babelsublr\bbl@tempa}%
5441 \AddToHook{shipout/after}{%
5442 \let\thepage\bbl@save@thepage}}{}
5443 \IfBabelLayout{counters}%
5444 {\let\bbl@latinarabic=\@arabic
5445 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5446 \let\bbl@asciroman=\@roman
5447 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5448 \let\bbl@asciiRoman=\@Roman
5449 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5450 \fi % end if layout
5451 (/xetex|texet)

```

9.4. 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).

```

5452 (*texet)
5453 \def\bbl@provide@extra#1{%
5454 % == auto-select encoding ==
5455 \ifx\bbl@encoding@select@off\@empty\else
5456 \bbl@ifunset{\bbl@encoding@#1}%
5457 {\def\elt##1{,##1,}%
5458 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5459 \count@\z@
5460 \bbl@foreach\bbl@tempe{%
5461 \def\bbl@tempd{##1}% Save last declared
5462 \advance\count@\@ne}%
5463 \ifnum\count@>\@ne % (1)
5464 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5465 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5466 \bbl@replace\bbl@tempa{ },}%
5467 \global\bbl@csarg\let{encoding@#1}\@empty
5468 \bbl@xin@{,\bbl@tempd,},{,\bbl@tempa,}%
5469 \ifin\@else % if main encoding included in ini, do nothing
5470 \let\bbl@tempb\relax
5471 \bbl@foreach\bbl@tempa{%
5472 \ifx\bbl@tempb\relax
5473 \bbl@xin@{,##1,},{,\bbl@tempe,}%
5474 \ifin\def\bbl@tempb{##1}\fi
5475 \fi}%
5476 \ifx\bbl@tempb\relax\else
5477 \bbl@exp{%
5478 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>%
5479 \gdef\<\bbl@encoding@#1>{%
5480 \\\babel@save\\f@encoding
5481 \\\bbl@add\\originalTeX{\\selectfont}%
5482 \\\fontencoding{\bbl@tempb}%
5483 \\\selectfont}}%
5484 \fi
5485 \fi
5486 \fi}%
5487 {}%
5488 \fi}

```

9.5. 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, `\bbl@hyphendata@<num>` exists (with the names of the files read).

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

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

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

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

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

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

```

5490 (*luatex)
5491 \directlua{ Babel = Babel or {} } % DL2
5492 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5493 \bbl@trace{Read language.dat}
5494 \ifx\bbl@readstream\undefined
5495   \csname newread\endcsname\bbl@readstream
5496 \fi
5497 \begingroup
5498   \toks@{}
5499   \count@ \z@ % 0=start, 1=0th, 2=normal
5500   \def\bbl@process@line#1#2 #3 #4 {%
5501     \ifx=#1%
5502       \bbl@process@synonym{#2}%
5503     \else
5504       \bbl@process@language{#1#2}{#3}{#4}%
5505     \fi
5506     \ignorespaces}
5507   \def\bbl@manylang{%
5508     \ifnum\bbl@last>\@ne
5509       \bbl@info{Non-standard hyphenation setup}%
5510     \fi
5511     \let\bbl@manylang\relax}
5512   \def\bbl@process@language#1#2#3{%
5513     \ifcase\count@
5514       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
5515     \or
5516       \count@\tw@

```

```

5517 \fi
5518 \ifnum\count@=\tw@
5519 \expandafter\addlanguage\csname l@#1\endcsname
5520 \language\allocationnumber
5521 \chardef\bbl@last\allocationnumber
5522 \bbl@manylang
5523 \let\bbl@elt\relax
5524 \xdef\bbl@languages{%
5525 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5526 \fi
5527 \the\toks@
5528 \toks@{}}
5529 \def\bbl@process@synonym@aux#1#2{%
5530 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5531 \let\bbl@elt\relax
5532 \xdef\bbl@languages{%
5533 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5534 \def\bbl@process@synonym#1{%
5535 \ifcase\count@
5536 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5537 \or
5538 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5539 \else
5540 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5541 \fi}
5542 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5543 \chardef\l@english\z@
5544 \chardef\l@USenglish\z@
5545 \chardef\bbl@last\z@
5546 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5547 \gdef\bbl@languages{%
5548 \bbl@elt{english}{0}{hyphen.tex}}%
5549 \bbl@elt{USenglish}{0}{}}
5550 \else
5551 \global\let\bbl@languages@format\bbl@languages
5552 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5553 \ifnum#2>\z@
5554 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5555 \fi}%
5556 \xdef\bbl@languages{\bbl@languages}%
5557 \fi
5558 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5559 \bbl@languages
5560 \openin\bbl@readstream=language.dat
5561 \ifeof\bbl@readstream
5562 \bbl@warning{I couldn't find language.dat. No additional\\%
5563 patterns loaded. Reported}%
5564 \else
5565 \loop
5566 \endlinechar\m@ne
5567 \read\bbl@readstream to \bbl@line
5568 \endlinechar\^^M
5569 \if T\ifeof\bbl@readstream F\fi T\relax
5570 \ifx\bbl@line\empty\else
5571 \edef\bbl@line{\bbl@line\space\space\space}%
5572 \expandafter\bbl@process@line\bbl@line\relax
5573 \fi
5574 \repeat
5575 \fi
5576 \closein\bbl@readstream
5577 \endgroup
5578 \bbl@trace{Macros for reading patterns files}
5579 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}

```

```

5580 \ifx\babelcatcodetablenum\@undefined
5581 \ifx\newcatcodetable\@undefined
5582 \def\babelcatcodetablenum{5211}
5583 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5584 \else
5585 \newcatcodetable\babelcatcodetablenum
5586 \newcatcodetable\bbl@pattcodes
5587 \fi
5588 \else
5589 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5590 \fi
5591 \def\bbl@luapatterns#1#2{%
5592 \bbl@get@enc#1::\@@@
5593 \setbox\z@\hbox\bgroup
5594 \begingroup
5595 \savecatcodetable\babelcatcodetablenum\relax
5596 \initcatcodetable\bbl@pattcodes\relax
5597 \catcodetable\bbl@pattcodes\relax
5598 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
5599 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5600 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5601 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5602 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5603 \catcode\`=12 \catcode\'=12 \catcode\"=12
5604 \input #1\relax
5605 \catcodetable\babelcatcodetablenum\relax
5606 \endgroup
5607 \def\bbl@tempa{#2}%
5608 \ifx\bbl@tempa\@empty\else
5609 \input #2\relax
5610 \fi
5611 \egroup}%
5612 \def\bbl@patterns@lua#1{%
5613 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5614 \csname l@#1\endcsname
5615 \edef\bbl@tempa{#1}%
5616 \else
5617 \csname l@#1:\f@encoding\endcsname
5618 \edef\bbl@tempa{#1:\f@encoding}%
5619 \fi\relax
5620 \namedef{lu@texhyphen@loaded@the\language}{% Temp
5621 \@ifundefined{bbl@hyphendata@the\language}%
5622 {\def\bbl@elt##1##2##3##4{%
5623 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5624 \def\bbl@tempb{##3}%
5625 \ifx\bbl@tempb\@empty\else % if not a synonymous
5626 \def\bbl@tempc{##3}{##4}%
5627 \fi
5628 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5629 \fi}%
5630 \bbl@languages
5631 \@ifundefined{bbl@hyphendata@the\language}%
5632 {\bbl@info{No hyphenation patterns were set for\%
5633 language '\bbl@tempa'. Reported}}%
5634 {\expandafter\expandafter\expandafter\bbl@luapatterns
5635 \csname bbl@hyphendata@the\language\endcsname}}}%
5636 \endinput\fi

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

5637 \ifx\DisableBabelHook\@undefined
5638 \AddBabelHook{luatex}{everylanguage}{%
5639 \def\process@language##1##2##3{%
5640 \def\process@line####1####2 ####3 ####4 {}}}

```

```

5641 \AddBabelHook{luatex}{loadpatterns}{%
5642   \input #1\relax
5643   \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5644     {{#1}}}}
5645 \AddBabelHook{luatex}{loadexceptions}{%
5646   \input #1\relax
5647   \def\bbl@tempb##1##2{{##1}{#1}}%
5648   \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5649     {\expandafter\expandafter\expandafter\bbl@tempb
5650       \csname bbl@hyphendata@the\language\endcsname}}
5651 \endinput\fi

```

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

```

5652 \begingroup
5653 \catcode`\%=12
5654 \catcode`\'=12
5655 \catcode`\ "=12
5656 \catcode`\:=12
5657 \directlua{
5658   Babel.locale_props = Babel.locale_props or {}
5659
5660   function Babel.lua_error(e, a)
5661     tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5662       e .. '}' .. (a or '') .. '}'})
5663   end
5664
5665   function Babel.bytes(line)
5666     return line:gsub(".",
5667       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5668   end
5669
5670   function Babel.priority_in_callback(name,description)
5671     for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5672       if v == description then return i end
5673     end
5674     return false
5675   end
5676
5677   function Babel.begin_process_input()
5678     if luatexbase and luatexbase.add_to_callback then
5679       luatexbase.add_to_callback('process_input_buffer',
5680         Babel.bytes,'Babel.bytes')
5681     else
5682       Babel.callback = callback.find('process_input_buffer')
5683       callback.register('process_input_buffer',Babel.bytes)
5684     end
5685   end
5686   function Babel.end_process_input ()
5687     if luatexbase and luatexbase.remove_from_callback then
5688       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5689     else
5690       callback.register('process_input_buffer',Babel.callback)
5691     end
5692   end
5693
5694   function Babel.str_to_nodes(fn, matches, base)
5695     local n, head, last
5696     if fn == nil then return nil end
5697     for s in string.utfvalues(fn(matches)) do
5698       if base.id == 7 then
5699         base = base.replace
5700       end

```

```

5701     n = node.copy(base)
5702     n.char    = s
5703     if not head then
5704         head = n
5705     else
5706         last.next = n
5707     end
5708     last = n
5709 end
5710 return head
5711 end
5712
5713 Babel.linebreaking = Babel.linebreaking or {}
5714 Babel.linebreaking.before = {}
5715 Babel.linebreaking.after = {}
5716 Babel.locale = {}
5717 function Babel.linebreaking.add_before(func, pos)
5718     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5719     if pos == nil then
5720         table.insert(Babel.linebreaking.before, func)
5721     else
5722         table.insert(Babel.linebreaking.before, pos, func)
5723     end
5724 end
5725 function Babel.linebreaking.add_after(func)
5726     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5727     table.insert(Babel.linebreaking.after, func)
5728 end
5729
5730 function Babel.addpatterns(pp, lg)
5731     local lg = lang.new(lg)
5732     local pats = lang.patterns(lg) or ''
5733     lang.clear_patterns(lg)
5734     for p in pp:gmatch('[^%s]+') do
5735         ss = ''
5736         for i in string.utfcharacters(p:gsub('%d', '')) do
5737             ss = ss .. '%d?' .. i
5738         end
5739         ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5740         ss = ss:gsub('%.%d%?$', '%%.')
5741         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5742         if n == 0 then
5743             tex.sprint(
5744                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5745                 .. p .. [[{ }]])
5746             pats = pats .. ' ' .. p
5747         else
5748             tex.sprint(
5749                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5750                 .. p .. [[{ }]])
5751         end
5752     end
5753     lang.patterns(lg, pats)
5754 end
5755
5756 Babel.characters = Babel.characters or {}
5757 Babel.ranges = Babel.ranges or {}
5758 function Babel.hlist_has_bidi(head)
5759     local has_bidi = false
5760     local ranges = Babel.ranges
5761     for item in node.traverse(head) do
5762         if item.id == node.id'glyph' then
5763             local itemchar = item.char

```

```

5764     local chardata = Babel.characters[itemchar]
5765     local dir = chardata and chardata.d or nil
5766     if not dir then
5767         for nn, et in ipairs(ranges) do
5768             if itemchar < et[1] then
5769                 break
5770             elseif itemchar <= et[2] then
5771                 dir = et[3]
5772                 break
5773             end
5774         end
5775     end
5776     if dir and (dir == 'al' or dir == 'r') then
5777         has_bidi = true
5778     end
5779 end
5780 end
5781 return has_bidi
5782 end
5783 function Babel.set_chrnges_b (script, chrng)
5784     if chrng == '' then return end
5785     texio.write('Package babel Info: Replacing ' .. script .. ' script ranges')
5786     Babel.script_blocks[script] = {}
5787     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5788         table.insert(
5789             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5790     end
5791 end
5792
5793 function Babel.discard_sublr(str)
5794     if str:find( [[\string\indexentry]] ) and
5795        str:find( [[\string\babelsublr]] ) then
5796         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5797             function(m) return m:sub(2,-2) end )
5798     end
5799     return str
5800 end
5801 }
5802 \endgroup
5803 \ifx\newattribute\@undefined\else % Test for plain
5804     \newattribute\bbl@attr@locale % DL4
5805     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5806     \AddBabelHook{luatex}{beforeextras}{%
5807         \setattribute\bbl@attr@locale\localeid}
5808 \fi
5809 %
5810 \def\BabelStringsDefault{unicode}
5811 \let\luabbl@stop\relax
5812 \AddBabelHook{luatex}{encodedcommands}{%
5813     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5814     \ifx\bbl@tempa\bbl@tempb\else
5815         \directlua{Babel.begin_process_input()}%
5816         \def\luabbl@stop{%
5817             \directlua{Babel.end_process_input()}}%
5818     \fi}%
5819 \AddBabelHook{luatex}{stopcommands}{%
5820     \luabbl@stop
5821     \let\luabbl@stop\relax}
5822 %
5823 \AddBabelHook{luatex}{patterns}{%
5824     \@ifundefined{bbl@hyphendata@the\language}%
5825     {\def\bbl@elt##1##2##3##4{%
5826         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...

```

```

5827 \def\bbl@tempb{##3}%
5828 \ifx\bbl@tempb\@empty\else % if not a synonymous
5829 \def\bbl@tempc{##3}{##4}%
5830 \fi
5831 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5832 \fi}%
5833 \bbl@languages
5834 \@ifundefined{bbl@hyphendata@the\language}%
5835 {\bbl@info{No hyphenation patterns were set for\%
5836 language '#2'. Reported}}%
5837 {\expandafter\expandafter\expandafter\bbl@luapatterns
5838 \csname bbl@hyphendata@the\language\endcsname}}}%
5839 \@ifundefined{bbl@patterns@}{}%
5840 \begingroup
5841 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5842 \ifin@else
5843 \ifx\bbl@patterns@\@empty\else
5844 \directlua{ Babel.addpatterns(
5845 [[\bbl@patterns@]], \number\language) }%
5846 \fi
5847 \@ifundefined{bbl@patterns@#1}%
5848 \@empty
5849 {\directlua{ Babel.addpatterns(
5850 [[\space\csname bbl@patterns@#1\endcsname]],
5851 \number\language) }}%
5852 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5853 \fi
5854 \endgroup}%
5855 \bbl@exp{%
5856 \bbl@ifunset{bbl@prehc@\languagename}}}%
5857 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5858 {\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.

```

5859 \@onlypreamble\babelpatterns
5860 \AtEndOfPackage{%
5861 \newcommand\babelpatterns[2][\@empty]{%
5862 \ifx\bbl@patterns@\relax
5863 \let\bbl@patterns@\@empty
5864 \fi
5865 \ifx\bbl@pttnlist@\@empty\else
5866 \bbl@warning{%
5867 You must not intermingle \string\selectlanguage\space and\%
5868 \string\babelpatterns\space or some patterns will not\%
5869 be taken into account. Reported}%
5870 \fi
5871 \ifx@\@empty#1%
5872 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5873 \else
5874 \edef\bbl@tempb{\zap@space#1 \@empty}%
5875 \bbl@for\bbl@tempa\bbl@tempb{%
5876 \bbl@fixname\bbl@tempa
5877 \bbl@iflanguage\bbl@tempa{%
5878 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5879 \@ifundefined{bbl@patterns@\bbl@tempa}%
5880 \@empty
5881 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5882 #2}}}%
5883 \fi}}

```

9.6. 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.

```
5884 \def\bbl@intraspace#1 #2 #3\@@{%
5885   \directlua{
5886     Babel.intraspaces = Babel.intraspaces or {}
5887     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5888       {b = #1, p = #2, m = #3}
5889     Babel.locale_props[\the\localeid].intraspace = %
5890       {b = #1, p = #2, m = #3}
5891   }}
5892 \def\bbl@intrapenalty#1\@@{%
5893   \directlua{
5894     Babel.intrapenalties = Babel.intrapenalties or {}
5895     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5896     Babel.locale_props[\the\localeid].intrapenalty = #1
5897   }}
5898 \begingroup
5899 \catcode`\%=12
5900 \catcode`\&=14
5901 \catcode`\'=12
5902 \catcode`\-=12
5903 \gdef\bbl@seaintraspace{&
5904   \let\bbl@seaintraspace\relax
5905   \directlua{
5906     Babel.sea_enabled = true
5907     Babel.sea_ranges = Babel.sea_ranges or {}
5908     function Babel.set_chranges (script, chrng)
5909       local c = 0
5910       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5911         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5912         c = c + 1
5913       end
5914     end
5915     function Babel.sea_disc_to_space (head)
5916       local sea_ranges = Babel.sea_ranges
5917       local last_char = nil
5918       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5919       local item, next_item = head
5920       while item do
5921         next_item = item.next
5922         local i = item.id
5923         if i == node.id'glyph' then
5924           last_char = item
5925         elseif i == 7 and item.subtype == 3 and last_char
5926           and last_char.char > 0x0C99 then
5927           quad = font.getfont(last_char.font).size
5928           for lg, rg in pairs(sea_ranges) do
5929             if last_char.char > rg[1] and last_char.char < rg[2] then
5930               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5931               local intraspace = Babel.intraspaces[lg]
5932               local intrapenalty = Babel.intrapenalties[lg]
5933               local n
5934               if intrapenalty ~= 0 then
5935                 n = node.new(14, 0)      &% penalty
5936                 n.penalty = intrapenalty
5937                 node.insert_before(head, item, n)
5938               end
5939               n = node.new(12, 13)      &% (glue, spaceskip)
5940               node.setglue(n, intraspace.b * quad,
```

```

5941             intraspace.p * quad,
5942             intraspace.m * quad)
5943         node.insert_before(head, item, n)
5944         node.remove(head, item)
5945         node.free(item)
5946         break
5947     end
5948 end
5949 end
5950 item = next_item
5951 end
5952 end
5953 }&
5954 \bbl@luahyphenate}

```

9.7. 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.

```

5955 \catcode`\%=14
5956 \gdef\bbl@cjk intraspace{%
5957   \let\bbl@cjk intraspace\relax
5958   \directlua{
5959     require('babel-data-cjk.lua')
5960     Babel.cjk_enabled = true
5961     function Babel.cjk_linebreak(head)
5962       local GLYPH = node.id'glyph'
5963       local last_char = nil
5964       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5965       local last_class = nil
5966       local last_lang = nil
5967       for item in node.traverse(head) do
5968         if item.id == GLYPH then
5969           local lang = item.lang
5970           local LOCALE = node.get_attribute(item,
5971             Babel.attr_locale)
5972           local props = Babel.locale_props[LOCALE] or {}
5973           local class = Babel.cjk_class[item.char].c
5974           if props.cjk_quotes and props.cjk_quotes[item.char] then
5975             class = props.cjk_quotes[item.char]
5976           end
5977           if class == 'cp' then class = 'cl' % ) as CL
5978           elseif class == 'id' then class = 'I'
5979           elseif class == 'cj' then class = 'I' % loose
5980           end
5981           local br = 0
5982           if class and last_class and Babel.cjk_breaks[last_class][class] then
5983             br = Babel.cjk_breaks[last_class][class]
5984           end
5985           if br == 1 and props.linebreak == 'c' and
5986             lang ~= \the\l@nohyphenation\space and
5987             last_lang ~= \the\l@nohyphenation then
5988             local intrapenalty = props.intrapenalty
5989             if intrapenalty ~= 0 then
5990               local n = node.new(14, 0)      % penalty
5991               n.penalty = intrapenalty
5992               node.insert_before(head, item, n)
5993             end
5994             local intraspace = props.intraspace

```

```

5995         local n = node.new(12, 13)      % (glue, spaceskip)
5996         node.setglue(n, intraspace.b * quad,
5997                     intraspace.p * quad,
5998                     intraspace.m * quad)
5999         node.insert_before(head, item, n)
6000     end
6001     if font.getfont(item.font) then
6002         quad = font.getfont(item.font).size
6003     end
6004     last_class = class
6005     last_lang = lang
6006     else % if penalty, glue or anything else
6007         last_class = nil
6008     end
6009 end
6010 lang.hyphenate(head)
6011 end
6012 }%
6013 \bbl@luahyphenate}
6014 \gdef\bbl@luahyphenate{%
6015 \let\bbl@luahyphenate\relax
6016 \directlua{
6017     luatexbase.add_to_callback('hyphenate',
6018     function (head, tail)
6019         if Babel.protect_tail then
6020             Babel.protect_tail(head, tail)
6021         end
6022         if Babel.linebreaking.before then
6023             for k, func in ipairs(Babel.linebreaking.before) do
6024                 func(head)
6025             end
6026         end
6027         lang.hyphenate(head)
6028         if Babel.cjk_enabled then
6029             Babel.cjk_linebreak(head)
6030         end
6031         if Babel.linebreaking.after then
6032             for k, func in ipairs(Babel.linebreaking.after) do
6033                 func(head)
6034             end
6035         end
6036         if Babel.set_hboxed then
6037             Babel.set_hboxed(head)
6038         end
6039         if Babel.sea_enabled then
6040             Babel.sea_disc_to_space(head)
6041         end
6042     end,
6043     'Babel.hyphenate')
6044 }}
6045 \endgroup
6046 %
6047 \def\bbl@provide@intraspace{%
6048 \bbl@ifunset\bbl@intsp@{language}\language\endcsname\empty}%
6049 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\empty\else
6050 \bbl@xin@{/c}{/bbl@cl{lnbrk}}}%
6051 \ifin@ % cjk
6052 \bbl@cjkintraspace
6053 \directlua{
6054     Babel.locale_props = Babel.locale_props or {}
6055     Babel.locale_props[\the\localeid].linebreak = 'c'
6056 }%
6057 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%

```

```

6058      \ifx\bbbl@KVP@intrapenalty\@nnil
6059      \bbbl@intrapenalty0\@@
6060      \fi
6061      \else          % sea
6062      \bbbl@seaintraspace
6063      \bbbl@exp{\bbbl@intraspace\bbbl@cl{intsp}}\@@@%
6064      \directlua{
6065          Babel.sea_ranges = Babel.sea_ranges or {}
6066          Babel.set_chranges('\bbbl@cl{sbcpr}',
6067                          '\bbbl@cl{chrng}')
6068      }%
6069      \ifx\bbbl@KVP@intrapenalty\@nnil
6070      \bbbl@intrapenalty0\@@
6071      \fi
6072      \fi
6073      \fi
6074      \ifx\bbbl@KVP@intrapenalty\@nnil\else
6075      \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
6076      \fi}}

```

9.8. Arabic justification

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

```

6077 \ifnum\bbbl@bidimode>100 \ifnum\bbbl@bidimode<200
6078 \def\bbblar@chars{%
6079 0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
6080 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
6081 0640,0641,0642,0643,0644,0645,0646,0647,0649}
6082 \def\bbblar@elongated{%
6083 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
6084 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
6085 0649,064A}
6086 \beginngroup
6087 \catcode\_:=11 \catcode\`:=11
6088 \gdef\bbblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
6089 \endgroup
6090 \gdef\bbbl@arabicjust{%
6091 \let\bbbl@arabicjust\relax
6092 \newattribute\bbblar@kashida
6093 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bbblar@kashida' }%
6094 \bbblar@kashida=\z@
6095 \bbbl@patchfont{\bbbl@parsejalt}}%
6096 \directlua{
6097 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
6098 Babel.arabic.elong_map[\the\localeid] = {}
6099 luatexbase.add_to_callback('post_linebreak_filter',
6100 Babel.arabic.justify, 'Babel.arabic.justify')
6101 luatexbase.add_to_callback('hpack_filter',
6102 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
6103 }}%

```

Save both node lists to make replacement.

```

6104 \def\bbblar@fetchjalt#1#2#3#4{%
6105 \bbbl@exp{\bbbl@foreach{#1}}{%
6106 \bbbl@ifunset{bbblar@JE@##1}%
6107 {\setbox\z@\hbox{\texdir TRT ^^^200d\char"##1#2}}%
6108 {\setbox\z@\hbox{\texdir TRT ^^^200d\char"@nameuse{bbblar@JE@##1}#2}}%
6109 \directlua{%
6110 local last = nil
6111 for item in node.traverse(tex.box[0].head) do
6112 if item.id == node.id'glyph' and item.char > 0x600 and
6113 not (item.char == 0x200D) then

```

```

6114         last = item
6115     end
6116 end
6117 Babel.arabic.#3['##1#4'] = last.char
6118 }}}

```

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

```

6119 \gdef\bbl@parsejalt{%
6120   \ifx\addfontfeature\undefined\else
6121     \bbl@xin@{/e}/{\bbl@ccl{\lnbrk}}}%
6122   \ifin@
6123     \directlua{%
6124       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
6125         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
6126         tex.print([[string\cswname\space bbl@parsejalti\endcswname]])
6127       end
6128     }%
6129   \fi
6130 \fi}
6131 \gdef\bbl@parsejalti{%
6132   \begingroup
6133     \let\bbl@parsejalt\relax % To avoid infinite loop
6134     \edef\bbl@tempb{\fontid\font}%
6135     \bblar@nofswarn
6136     \@nameuse{tracinglostchars}\z@
6137     \bblar@fetchjalt\bblar@elongated{}{from}{}%
6138     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
6139     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
6140     \addfontfeature{RawFeature+=jalt}%
6141     % \@namedef{bblar@JE0643}{06AA}% todo: catch medial kaf
6142     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
6143     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
6144     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
6145     \directlua{%
6146       for k, v in pairs(Babel.arabic.from) do
6147         if Babel.arabic.dest[k] and
6148            not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
6149           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
6150             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
6151         end
6152       end
6153     }%
6154   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

6155 \begingroup
6156 \catcode`#=11
6157 \catcode`~=11
6158 \directlua{
6159
6160 Babel.arabic = Babel.arabic or {}
6161 Babel.arabic.from = {}
6162 Babel.arabic.dest = {}
6163 Babel.arabic.justify_factor = 0.95
6164 Babel.arabic.justify_enabled = true
6165 Babel.arabic.kashida_limit = -1
6166
6167 function Babel.arabic.justify(head)
6168   if not Babel.arabic.justify_enabled then return head end
6169   local HLIST = node.id'hlist'
6170   local line, next_line = head
6171   while line do
6172     next_line = line.next

```

```

6173     if line.id == HLIST then
6174         % The line may be replaced (and freed), even the first one (eg,
6175         % in \vbox), so the head is returned.
6176         head = Babel.arabic.justify_hlist(head, line)
6177     end
6178     line = next_line
6179 end
6180 return head
6181 end
6182
6183 function Babel.arabic.justify_hbox(head, gc, size, pack)
6184     local has_inf = false
6185     if Babel.arabic.justify_enabled and pack == 'exactly' then
6186         for n in node.traverse_id(12, head) do
6187             if n.stretch_order > 0 then has_inf = true end
6188         end
6189         if not has_inf then
6190             head = Babel.arabic.justify_hlist(head, nil, gc, size, pack)
6191         end
6192     end
6193     return head
6194 end
6195
6196 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
6197     local d, new
6198     local k_list, k_item, pos_inline
6199     local width, width_new, full, k_curr, wt_pos, goal, shift
6200     local subst_done = false
6201     local elong_map = Babel.arabic.elong_map
6202     local cnt
6203     local last_line
6204     local GLYPH = node.id'glyph'
6205     local KASHIDA = Babel.attr_kashida
6206     local LOCALE = Babel.attr_locale
6207
6208     if line == nil then
6209         line = {}
6210         line.glue_sign = 1
6211         line.glue_order = 0
6212         line.head = head
6213         line.shift = 0
6214         line.width = size
6215     end
6216
6217     % Exclude last line.
6218     if ((line.next ~= nil or line.glue_sign == 1) and line.glue_order == 0) then
6219         elongs = {}      % Stores elongated candidates of each line
6220         k_list = {}      % And all letters with kashida
6221         pos_inline = 0    % Not yet used
6222
6223         for n in node.traverse_id(GLYPH, line.head) do
6224             pos_inline = pos_inline + 1 % To find where it is. Not used.
6225
6226             % Elongated glyphs
6227             if elong_map then
6228                 local locale = node.get_attribute(n, LOCALE)
6229                 if elong_map[locale] and elong_map[locale][n.font] and
6230                     elong_map[locale][n.font][n.char] then
6231                     table.insert(elongs, {node = n, locale = locale} )
6232                     node.set_attribute(n, KASHIDA, 0)
6233                 end
6234             end
6235

```

```

6236 % Tatwil. First create a list of nodes marked with kashida. The
6237 % rest of nodes can be ignored. The list of used weights is build
6238 % when transforms with the key kashida= are declared.
6239 if Babel.kashida_wts then
6240     local k_wt = node.get_attribute(n, KASHIDA)
6241     if k_wt > 0 then % todo. parameter for multi inserts
6242         table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
6243     end
6244 end
6245
6246 end % of node.traverse_id
6247
6248 if #elongs == 0 and #k_list == 0 then goto next_line end
6249 full = line.width
6250 shift = line.shift
6251 goal = full * Babel.arabic.justify_factor % A bit crude
6252 width = node.dimensions(line.head) % The 'natural' width
6253
6254 % == Elongated ==
6255 % Original idea taken from 'chickenize'
6256 while (#elongs > 0 and width < goal) do
6257     subst_done = true
6258     local x = #elongs
6259     local curr = elongs[x].node
6260     local oldchar = curr.char
6261     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6262     width = node.dimensions(line.head) % Check if the line is too wide
6263     % Substitute back if the line would be too wide and break:
6264     if width > goal then
6265         curr.char = oldchar
6266         break
6267     end
6268     % If continue, pop the just substituted node from the list:
6269     table.remove(elongs, x)
6270 end
6271
6272 % == Tatwil ==
6273 % Traverse the kashida node list so many times as required, until
6274 % the line is filled. The first pass adds a tatweel after each
6275 % node with kashida in the line, the second pass adds another one,
6276 % and so on. In each pass, add first the kashida with the highest
6277 % weight, then with lower weight and so on.
6278 if #k_list == 0 then goto next_line end
6279
6280 width = node.dimensions(line.head) % The 'natural' width
6281 k_curr = #k_list % Traverse backwards, from the end
6282 wt_pos = 1
6283
6284 while width < goal do
6285     subst_done = true
6286     k_item = k_list[k_curr].node
6287     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6288         d = node.copy(k_item)
6289         d.char = 0x0640
6290         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6291         d.xoffset = 0
6292         line.head, new = node.insert_after(line.head, k_item, d)
6293         width_new = node.dimensions(line.head)
6294         if width > goal or width == width_new then
6295             node.remove(line.head, new) % Better compute before
6296             node.free(new)
6297             break
6298         end

```

```

6299         if Babel.fix_diacr then
6300             Babel.fix_diacr(k_item.next)
6301         end
6302         width = width_new
6303     end
6304     if k_curr == 1 then
6305         k_curr = #k_list
6306         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6307     else
6308         k_curr = k_curr - 1
6309     end
6310 end
6311
6312 % Limit the number of tatweel by removing them. Not very efficient,
6313 % but it does the job in a quite predictable way.
6314 if Babel.arabic.kashida_limit > -1 then
6315     cnt = 0
6316     local n, next_n = line.head
6317     while n do
6318         next_n = n.next
6319         if n.id == GLYPH and n.char == 0x0640 then
6320             cnt = cnt + 1
6321             if cnt > Babel.arabic.kashida_limit then
6322                 line.head = node.remove(line.head, n)
6323                 node.free(n)
6324             end
6325         elseif n.id == GLYPH then
6326             cnt = 0
6327         end
6328         n = next_n
6329     end
6330 end
6331
6332 ::next_line::
6333
6334 % Must take into account marks and ins, see luatex manual.
6335 % Have to be executed only if there are changes. Investigate
6336 % what's going on exactly.
6337 if subst_done and not gc then
6338     d = node.hpack(line.head, full, 'exactly')
6339     d.shift = shift
6340     head = node.insert_before(head, line, d)
6341     head = node.remove(head, line)
6342     line.head = nil % Now in d
6343     node.free(line)
6344 end
6345 end % if process line
6346 % The list of a box (line is a table) may start with a removed tatweel
6347 return (line.id and head) or line.head
6348 end
6349 }
6350 \endgroup
6351 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

9.9. 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`.

```

6352 \def\bbl@scr@node@list{%
6353   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%

```

```

6354 ,Greek, Latin, Old Church Slavonic Cyrillic,}
6355 \ifnum\bbl@bidimode=102 % bidi-r
6356   \bbl@add\bbl@scr@node@list{Arabic, Hebrew, Syriac}
6357 \fi
6358 \def\bbl@set@renderer{%
6359   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6360   \ifin@
6361     \let\bbl@unset@renderer\relax
6362   \else
6363     \bbl@exp{%
6364       \def\\bbl@unset@renderer{%
6365         \def<g__fontspec_default_fontopts_clist>{%
6366           \[g__fontspec_default_fontopts_clist]}}%
6367       \def<g__fontspec_default_fontopts_clist>{%
6368         Renderer=Harfbuzz, \[g__fontspec_default_fontopts_clist]}}%
6369     \fi}
6370 <@Font selection@>

```

9.10. 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.

There are two situations where the replacement is not carried out: either the `letters` option has been set and the character is not a letter (in the $\TeX$ sense), or the current script is the same as the new one.

```

6371 \directlua{% DL6
6372 Babel.script_blocks = {
6373   ['dflt'] = {},
6374   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6375               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6376   ['Armn'] = {{0x0530, 0x058F}},
6377   ['Beng'] = {{0x0980, 0x09FF}},
6378   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6379   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6380   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6381              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6382   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6383   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6384              {0xAB00, 0xAB2F}},
6385   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6386   % Don't follow strictly Unicode, which places some Coptic letters in
6387   % the 'Greek and Coptic' block
6388   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6389   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6390               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6391               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6392               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6393               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6394               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6395   ['Hebr'] = {{0x0590, 0x05FF},
6396               {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6397   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6398               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6399   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6400   ['Knda'] = {{0x0C80, 0x0CFF}},
6401   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6402               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF}},

```

```

6403         {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6404 ['Lao'] = {{0xE80, 0xEFF}},
6405 ['Latn'] = {{0x000, 0x07F}, {0x080, 0x0FF}, {0x100, 0x17F},
6406         {0x180, 0x24F}, {0x1E0, 0x1EF}, {0x2C0, 0x2CF},
6407         {0xA70, 0xA7F}, {0xAB30, 0xAB6F}},
6408 ['Mahj'] = {{0x1150, 0x117F}},
6409 ['Mlym'] = {{0x0D00, 0x0D7F}},
6410 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6411 ['Orya'] = {{0x0B00, 0x0B7F}},
6412 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x11E0, 0x11FF}},
6413 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6414 ['Taml'] = {{0x0B80, 0x0BFF}},
6415 ['Telu'] = {{0x0C00, 0x0C7F}},
6416 ['Tfng'] = {{0x2D30, 0x2D7F}},
6417 ['Thai'] = {{0x0E00, 0x0E7F}},
6418 ['Tibt'] = {{0x0F00, 0x0FFF}},
6419 ['Vaii'] = {{0xA500, 0xA63F}},
6420 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6421 }
6422
6423 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6424 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6425 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6426
6427 function Babel.locale_map(head)
6428   if not Babel.locale_mapped then return head end
6429
6430   local LOCALE = Babel.attr_locale
6431   local GLYPH = node.id('glyph')
6432   local inmath = false
6433   local toloc_save
6434   for item in node.traverse(head) do
6435     local toloc
6436     if not inmath and item.id == GLYPH then
6437       % Optimization: build a table with the chars found
6438       if Babel.chr_to_loc[item.char] then
6439         toloc = Babel.chr_to_loc[item.char]
6440       else
6441         for lc, maps in pairs(Babel.loc_to_scr) do
6442           for _, rg in pairs(maps) do
6443             if item.char >= rg[1] and item.char <= rg[2] then
6444               Babel.chr_to_loc[item.char] = lc
6445               toloc = lc
6446               break
6447             end
6448           end
6449         end
6450         % Treat composite chars in a different fashion, because they
6451         % 'inherit' the previous locale.
6452         if (item.char >= 0x0300 and item.char <= 0x036F) or
6453            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6454            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6455           Babel.chr_to_loc[item.char] = -2000
6456           toloc = -2000
6457         end
6458         if not toloc then
6459           Babel.chr_to_loc[item.char] = -1000
6460         end
6461       end
6462       if toloc == -2000 then
6463         toloc = toloc_save
6464       elseif toloc == -1000 then
6465         toloc = nil

```

```

6466     end
6467     if toloc and Babel.locale_props[toloc] and
6468         Babel.locale_props[toloc].letters and
6469         tex.getcatcode(item.char) \string~= 11 then
6470         toloc = nil
6471     end
6472     if toloc and Babel.locale_props[toloc].script
6473         and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6474         and Babel.locale_props[toloc].script ==
6475         Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6476         toloc = nil
6477     end
6478     if toloc then
6479         if Babel.locale_props[toloc].lg then
6480             item.lang = Babel.locale_props[toloc].lg
6481             node.set_attribute(item, LOCALE, toloc)
6482         end
6483         if Babel.locale_props[toloc]['/'..item.font] then
6484             item.font = Babel.locale_props[toloc]['/'..item.font]
6485         end
6486     end
6487     toloc_save = toloc
6488     elseif not inmath and item.id == 7 then % Apply recursively
6489         item.replace = item.replace and Babel.locale_map(item.replace)
6490         item.pre      = item.pre and Babel.locale_map(item.pre)
6491         item.post      = item.post and Babel.locale_map(item.post)
6492     elseif item.id == node.id'math' then
6493         inmath = (item.subtype == 0)
6494     end
6495 end
6496 return head
6497 end
6498 }

```

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

```

6499 \newcommand\babelcharproperty[1]{%
6500   \count@=#1\relax
6501   \ifvmode
6502     \expandafter\bbl@chprop
6503   \else
6504     \bbl@error{charproperty-only-vertical}{}{}{}%
6505   \fi}
6506 \newcommand\bbl@chprop[3][\the\count@]{%
6507   \@tempcnta=#1\relax
6508   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6509   {\bbl@error{unknown-char-property}{}{#2}{}}%
6510   {}%
6511   \loop
6512     \bbl@cs{chprop@#2}{#3}%
6513     \ifnum\count@<\@tempcnta
6514       \advance\count@\@ne
6515     \repeat}
6516 %
6517 \def\bbl@chprop@direction#1{%
6518   \directlua{
6519     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6520     Babel.characters[\the\count@]['d'] = '#1'
6521   }}
6522 \let\bbl@chprop@bc\bbl@chprop@direction
6523 %
6524 \def\bbl@chprop@mirror#1{%
6525   \directlua{

```

```

6526 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6527 Babel.characters[\the\count@]['m'] = '\number#1'
6528 }}
6529 \let\bbl@chprop@bmg\bbl@chprop@mirror
6530 %
6531 \def\bbl@chprop@linebreak#1{%
6532 \directlua{
6533 Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6534 Babel.cjk_characters[\the\count@]['c'] = '#1'
6535 }}
6536 \let\bbl@chprop@lb\bbl@chprop@linebreak
6537 %
6538 \def\bbl@chprop@locale#1{%
6539 \directlua{
6540 Babel.chr_to_loc = Babel.chr_to_loc or {}
6541 Babel.chr_to_loc[\the\count@] =
6542 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6543 }}

```

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.

```

6544 \directlua{% DL7
6545 Babel.nohyphenation = \the\l@nohyphenation
6546 }

```

Now the $\TeX$ high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the $\{n\}$ syntax. For example, $\text{pre}=\{1\}\{1\}$ becomes $\text{function}(m) \text{ return } m[1]..m[1]..'-'$ end, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m) \text{ return } \text{Babel.capt_map}(m[1], 1)$ 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 $\text{\directlua}$ does not take into account the current catcode of $@$, we just avoid this character in macro names (which explains the internal group, too).

```

6547 \begingroup
6548 \catcode`\-=12
6549 \catcode`\%=12
6550 \catcode`\&=14
6551 \catcode`\|=12
6552 \gdef\babelprehyphenation{%&
6553 \@@ifnextchar{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
6554 \gdef\babelposthyphenation{%&
6555 \@@ifnextchar{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
6556 %
6557 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6558 \ifcase#1
6559 \bbl@activateprehyphen
6560 \or
6561 \bbl@activateposthyphen
6562 \fi
6563 \begingroup
6564 \def\babeltempa{\bbl@add@list\babeltempb}%&
6565 \let\babeltempb\@empty
6566 \def\bbl@tempa{#5}%&
6567 \bbl@replace\bbl@tempa{,}{ ,}%& TODO. Ugly trick to preserve {}
6568 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6569 \bbl@ifsamestring{##1}{remove}%&
6570 {\bbl@add@list\babeltempb{nil}}}%&
6571 {\directlua{
6572 local rep = [=##1]=]
6573 local three_args = '%s*=%s*([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)'
6574 & Numeric passes directly: kern, penalty...
6575 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6576 rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')

```

```

6577     rep = rep:gsub('^%s*(after)%s*',',', 'after = true, ')
6578     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6579     rep = rep:gsub('node%s*=%s*([a+)%s*([a+])', Babel.capture_node)
6580     rep = rep:gsub(' (norule)' .. three_args,
6581       'norule = {' .. '%2, %3, %4' .. '}')
6582   if #1 == 0 or #1 == 2 then
6583     rep = rep:gsub(' (space)' .. three_args,
6584       'space = {' .. '%2, %3, %4' .. '}')
6585     rep = rep:gsub(' (spacefactor)' .. three_args,
6586       'spacefactor = {' .. '%2, %3, %4' .. '}')
6587     rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6588     &% Transform values
6589     rep, n = rep:gsub( '{([%a%-%.]+)|([%a%-%.]+)}',
6590       function(v,d)
6591         return string.format (
6592           '\the\csname bbl@id@@#3\endcsname,"%s",%s',
6593           v,
6594           load( 'return Babel.locale_props'..
6595             '\the\csname bbl@id@@#3\endcsname].' .. d)() )
6596       end )
6597     rep, n = rep:gsub( '{([%a%-%.]+)|([%-d%.]+)}',
6598       '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6599   end
6600   if #1 == 1 then
6601     rep = rep:gsub( ' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6602     rep = rep:gsub( ' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6603     rep = rep:gsub( ' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6604   end
6605   tex.print([[\\string\babeltempa{[]] .. rep .. [[]]])
6606   }}&%
6607 \bbl@foreach\babeltempb{&%
6608   \bbl@forkv{##1}}{&%
6609     \in@{,###1,},{,nil,step,data,remove,insert,string,no,pre,no,&%
6610       post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6611     \ifin@else
6612       \bbl@error{bad-transform-option}{###1}{}}{&%
6613     \fi}}&%
6614 \let\bbl@kv@attribute\relax
6615 \let\bbl@kv@label\relax
6616 \let\bbl@kv@fonts@empty
6617 \let\bbl@kv@prepend\relax
6618 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6619 \ifx\bbl@kv@fonts@empty\else\bbl@settransfont\fi
6620 \ifx\bbl@kv@attribute\relax
6621   \ifx\bbl@kv@label\relax\else
6622     \bbl@exp{\\bbl@trim@def\\bbl@kv@fonts{\bbl@kv@fonts}}&%
6623     \bbl@replace\bbl@kv@fonts{ }{,}&%
6624     \edef\bbl@kv@attribute{\bbl@ATR@bbl@kv@label @#3@bbl@kv@fonts}&%
6625     \count@ \z@
6626     \def\bbl@elt##1##2##3{&%
6627       \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6628       {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6629         {\count@ \@ne}&%
6630         {\bbl@error{font-conflict-transforms}{}}{}}}&%
6631       }}&%
6632   \bbl@transfont@list
6633   \ifnum\count@=\z@
6634     \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6635       {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6636   \fi
6637 \bbl@ifunset{\bbl@kv@attribute}&%
6638 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6639 {}&%

```

```

6640     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6641     \fi
6642   \else
6643     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6644   \fi
6645   \directlua{
6646     local lbkr = Babel.linebreaking.replacements[#1]
6647     local u = unicode.utf8
6648     local id, attr, label
6649     if #1 == 0 then
6650       id = \the\csname bbl@id@@#3\endcsname\space
6651     else
6652       id = \the\csname l@#3\endcsname\space
6653     end
6654     \ifx\bbl@kv@attribute\relax
6655       attr = -1
6656     \else
6657       attr = luatexbase.registernumber'\bbl@kv@attribute'
6658     \fi
6659     \ifx\bbl@kv@label\relax\else &% Same refs:
6660       label = [==[\bbl@kv@label]==]
6661     \fi
6662     &% Convert pattern:
6663     local patt = string.gsub([==[#4]==], '%s', '')
6664     if #1 == 0 then
6665       patt = string.gsub(patt, '|', ' ')
6666     end
6667     if not u.find(patt, '()', nil, true) then
6668       patt = '()' .. patt .. '()'
6669     end
6670     patt = string.gsub(patt, '%(%)%^', '^()')
6671     patt = string.gsub(patt, '%$%(%)', '()$')
6672     patt = u.gsub(patt, '{(.)}',
6673       function (n)
6674         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6675       end)
6676     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6677       function (n)
6678         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6679       end)
6680     lbkr[id] = lbkr[id] or {}
6681     table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6682       { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6683   }&%
6684   \endgroup}
6685 \endgroup
6686 %
6687 \let\bbl@transfont@list\@empty
6688 \def\bbl@settransfont{%
6689   \global\let\bbl@settransfont\relax % Execute only once
6690   \gdef\bbl@transfont{%
6691     \def\bbl@elt####1####2####3{%
6692       \bbl@ifblank{####3}%
6693       {\count@tw}% Do nothing if no fonts
6694       {\count@z@
6695         \bbl@vforeach{####3}{%
6696           \def\bbl@tempd{#####1}%
6697           \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6698           \ifx\bbl@tempd\bbl@tempe
6699             \count@\@ne
6700           \else\ifx\bbl@tempd\bbl@transfam
6701             \count@\@ne
6702           \fi\fi}%

```

```

6703         \ifcase\count@
6704         \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
6705         \or
6706         \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
6707         \fi}%
6708         \bbl@transfont@list}%
6709 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6710 \gdef\bbl@transfam{-unknown-}%
6711 \bbl@foreach\bbl@font@fams{%
6712   \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6713   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6714   {\xdef\bbl@transfam{##1}}%
6715   {}}
6716 %
6717 \DeclareRobustCommand\enablelocaletransform[1]{%
6718   \bbl@ifunset{bbl@ATR@#1@\language @}%
6719   {\bbl@error{transform-not-available}{#1}{}}%
6720   {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6721 \DeclareRobustCommand\disablelocaletransform[1]{%
6722   \bbl@ifunset{bbl@ATR@#1@\language @}%
6723   {\bbl@error{transform-not-available-b}{#1}{}}%
6724   {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}

```

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

```

6725 \def\bbl@activateposthyphen{%
6726   \let\bbl@activateposthyphen\relax
6727   \ifx\bbl@attr@hboxed\undefined
6728     \newattribute\bbl@attr@hboxed
6729   \fi
6730   \directlua{
6731     require('babel-transforms.lua')
6732     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6733   }}
6734 \def\bbl@activateprehyphen{%
6735   \let\bbl@activateprehyphen\relax
6736   \ifx\bbl@attr@hboxed\undefined
6737     \newattribute\bbl@attr@hboxed
6738   \fi
6739   \directlua{
6740     require('babel-transforms.lua')
6741     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6742   }}
6743 \newcommand\SetTransformValue[3]{%
6744   \directlua{
6745     Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6746   }}

```

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.

```

6747 \newcommand\ShowBabelTransforms[1]{%
6748   \bbl@activateprehyphen
6749   \bbl@activateposthyphen
6750   \begingroup
6751     \directlua{ Babel.show_transforms = true }%
6752     \setbox\z@\vbox{#1}%
6753     \directlua{ Babel.show_transforms = false }%
6754   \endgroup}

```

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.

```
6755 \newcommand\localeprehyphenation[1]{%
6756 \directlua{ Babel.string_prehyphenation([==[#1]==], \the\localeid) }}
```

9.11. Bidi

Make sure the `\pagedir` is always the same as the `\bodydir` when the document starts. At this point the main language has been selected, and therefore its `\bodydir` has been set.

```
6757 \AtBeginDocument{\pagedir\bodydir}
```

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.

```
6758 \def\bbl@activate@preotf{%
6759 \let\bbl@activate@preotf\relax % only once
6760 \directlua{
6761   function Babel.pre_otfload_v(head)
6762     if Babel.numbers and Babel.digits_mapped then
6763       head = Babel.numbers(head)
6764     end
6765     if Babel.bidi_enabled then
6766       head = Babel.bidi(head, false, dir)
6767     end
6768     return head
6769   end
6770   %
6771   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6772     if Babel.numbers and Babel.digits_mapped then
6773       head = Babel.numbers(head)
6774     end
6775     if Babel.bidi_enabled then
6776       head = Babel.bidi(head, false, dir)
6777     end
6778     return head
6779   end
6780   %
6781   luatexbase.add_to_callback('pre_linebreak_filter',
6782     Babel.pre_otfload_v,
6783     'Babel.pre_otfload_v',
6784     Babel.priority_in_callback('pre_linebreak_filter',
6785       'luaotfload.node_processor') or nil)
6786   %
6787   luatexbase.add_to_callback('hpack_filter',
6788     Babel.pre_otfload_h,
6789     'Babel.pre_otfload_h',
6790     Babel.priority_in_callback('hpack_filter',
6791       'luaotfload.node_processor') or nil)
6792 }
```

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@insidemath` hack is activated (set to 1) 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`.

```
6793 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6794 \let\bbl@beforeforeign\leavevmode
6795 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6796 \ifnum\bbl@bidimode>300
6797   \RequirePackage{unibidi-lua}
6798 \else
6799   \RequirePackage{luatexbase}
6800   \bbl@activate@preotf
6801   \directlua{
```

```

6802     require('babel-data-bidi.lua')
6803     \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
6804         require('babel-bidi-basic.lua')
6805     \or
6806         require('babel-bidi-basic-r.lua')
6807         table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6808         table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6809         table.insert(Babel.ranges, {0x100000, 0x10FFFFD, 'on'})
6810     \fi}
6811 \fi
6812 \newattribute\bbbl@attr@dir
6813 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
6814 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
6815 \fi
6816 %
6817 \chardef\bbbl@thetextdir\z@
6818 \chardef\bbbl@thepardir\z@
6819 \def\bbbl@setluadir#1#2{% l=\text/pardirection 2= 0:l 1:r 2:al
6820     \ifcase#2\relax
6821         \ifcase#1\else#1=\z@\fi
6822     \else
6823         \ifcase#1#1=\@ne\fi
6824     \fi}

    \bbbl@attr@dir stores the directions with a mask: ..00PPTT, with masks 0xC (PP is the par dir) and
    0x3 (TT is the text dir). These macro names are shared by the 3 engines, with different definitions.

6825 \def\bbbl@thedir{0}
6826 \def\bbbl@textdir#1{%
6827     \bbbl@setluadir\textdirection{#1}%
6828     \chardef\bbbl@thetextdir#1\relax
6829     \edef\bbbl@thedir{\the\numexpr\bbbl@thepardir*4+#1}%
6830     \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*4+#1}}
6831 \def\bbbl@pardir#1{% Used twice
6832     \bbbl@setluadir\pardirection{#1}%
6833     \chardef\bbbl@thepardir#1\relax}
6834 \def\bbbl@bodydir{\bbbl@setluadir\bodydirection}% Used once
6835 \def\bbbl@dirparastext{\pardirection=\textdirection\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.

6836 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6837     \def\bbbl@insidemath{0}%
6838     \def\bbbl@everymath{\def\bbbl@insidemath{1}}
6839     \def\bbbl@everydisplay{\def\bbbl@insidemath{2}}
6840     \frozen@everymath\expandafter{%
6841         \expandafter\bbbl@everymath\the\bbbl@everymath}
6842     \frozen@everydisplay\expandafter{%
6843         \expandafter\bbbl@everydisplay\the\bbbl@everydisplay}
6844 \AtBeginDocument{
6845     \directlua{
6846         function Babel.math_box_dir(head)
6847             if not (token.get_macro('bbbl@insidemath') == '0') then
6848                 if Babel.hlist_has_bidi(head) then
6849                     local d = node.new(node.id'dir')
6850                     d.dir = '+TRT'
6851                     for item in node.traverse(head) do
6852                         if item.id == 11 or item.id == node.id'glyph' then
6853                             head = node.insert_before(head, item, d)
6854                             break
6855                         end
6856                     end
6857                     local inmath = false
6858                     for item in node.traverse(head) do
6859                         if item.id == 11 then

```

```

6860         inmath = (item.subtype == 0)
6861     elseif not inmath then
6862         node.set_attribute(item,
6863             Babel.attr_dir, token.get_macro('bbl@thedir'))
6864     end
6865 end
6866 end
6867 end
6868 return head
6869 end
6870 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6871     "Babel.math_box_dir", 0)
6872 if Babel.unset_atdir then
6873     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6874         "Babel.unset_atdir")
6875     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6876         "Babel.unset_atdir")
6877 end
6878 luatexbase.add_to_callback("post_mlist_to_hlist_filter", function(head)
6879     local dir = tex.mathdirection
6880     local n = node.new("dir")
6881     n.direction = dir
6882     head = node.insert_before(head, head, n)
6883     n = node.new("dir", 1)
6884     n.direction = dir
6885     head = node.insert_after(head, node.tail(head), n)
6886     return head
6887 end, "Babel.ensure_math_dir")
6888 } }%
6889 \fi
6890 \ifnum\bbl@bidimode>300
6891     \def\bbl@sublr#1{^^^2066#1^^^2069}
6892 \fi

Experimental. Tentative name.

6893 \DeclareRobustCommand\localebox[1]{%
6894     {\def\bbl@insidemath{0}%
6895         \mbox{\foreignlanguage{\language}{#1}}}}

```

9.12. Layout

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

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

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

6896 \bbl@trace{Redefinitions for bidi layout}
6897 %
6898 <(*More package options)> ▶

```

```

6899 \chardef\bbl@eqnpos\z@
6900 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6901 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6902 {/More package options})

```

Fixes in luatex are sometimes done with flags. We first activate all all them.

```

6903 \ifnum\bbl@bidimode>\z@ % Any bidi=
6904   \matheqdirmode\@ne
6905   \mathemptydisplaymode\@ne
6906   \breakafterdirmode\@ne
6907   \fixupboxesmode\@ne
6908   \let\bbl@eqnodir\relax
6909   \def\bbl@eqdel{()}
6910   \def\bbl@eqnum{%
6911     {\normalfont\normalcolor
6912       \expandafter\@firstoftwo\bbl@eqdel
6913       \theequation
6914       \expandafter\@secondoftwo\bbl@eqdel}}
6915   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6916   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6917   \def\bbl@eqno@flip#1{% So
6918     \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6919       \leqno
6920       \hb@xt@.01pt{%
6921         \hb@xt@\displaywidth{\hss#1\glet\bbl@upset\@currentlabel}}\hss}%
6922     \else
6923       \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6924     \fi
6925     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6926   \def\bbl@leqno@flip#1{%
6927     \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6928       \leqno
6929       \hb@xt@.01pt{%
6930         \hss\hb@xt@\displaywidth{\#1\glet\bbl@upset\@currentlabel}\hss}}%
6931     \else
6932       \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6933     \fi
6934     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6935 %
6936 \AtBeginDocument{%
6937   \ifx\bbl@noamsmath\relax\else
6938   \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6939     \ifnum\bbl@eqnpos=\tw@
6940       \bbl@replace\equation{\hb@xt@\linewidth}%
6941       {\hbox bdir\mathdirection to\linewidth}%
6942       \bbl@carg\bbl@sreplace{[ ]}{\hb@xt@\linewidth}%
6943       {\hbox bdir\mathdirection to\linewidth}%
6944     \fi
6945     \AddToHook{env/equation/begin}{%
6946       \ifnum\bbl@thetextdir>\z@
6947         \let\@eqnnum\bbl@eqnum
6948         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6949         \chardef\bbl@thetextdir\z@
6950         \bbl@add\normalfont{\bbl@eqnodir}%
6951         \ifcase\bbl@eqnpos
6952           \let\bbl@puteqno\bbl@eqno@flip
6953         \or
6954           \let\bbl@puteqno\bbl@leqno@flip
6955         \fi
6956       \fi}%
6957     \ifnum\bbl@eqnpos=\tw@\else
6958       \def\endequation{\bbl@puteqno{\@eqnnum}$\@ignoretrue}%
6959     \fi

```

```

6960 \AddToHook{env/eqnarray/begin}{%
6961 \ifnum\bbbl@thetextdir>\z@
6962 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6963 \chardef\bbbl@thetextdir\z@
6964 \bbbl@add\normalfont{\bbbl@eqnodir}%
6965 \ifnum\bbbl@eqnpos=\@ne
6966 \def\@eqnnum{%
6967 \setbox\z@\hbox{\bbbl@eqnum}%
6968 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6969 \else
6970 \let\@eqnnum\bbbl@eqnum
6971 \fi
6972 \fi}
6973 \else % amstex
6974 \bbbl@exp{% Hack to hide maybe undefined conditionals:
6975 \chardef\bbbl@eqnpos=0\<iftagsleft>1\<else>\<if@flegn>2\<fi>\<fi>\relax}%
6976 \ifnum\bbbl@eqnpos=\@ne
6977 \let\bbbl@ams@lap\hbox
6978 \else
6979 \let\bbbl@ams@lap\llap
6980 \fi
6981 \ExplSyntaxOn % Required by \bbbl@sreplace with \intertext@
6982 \bbbl@sreplace\intertext@{\normalbaselines}%
6983 {\normalbaselines
6984 \ifx\bbbl@eqnodir\relax\else\bbbl@pardir\@ne\bbbl@eqnodir\fi}%
6985 \ExplSyntaxOff
6986 \def\bbbl@ams@tagbox#1#2{\#1{\bbbl@eqnodir#2}}% #1=hbox|@lap|flip
6987 \ifx\bbbl@ams@lap\hbox % leqno
6988 \def\bbbl@ams@flip#1{%
6989 \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6990 \else % eqno
6991 \def\bbbl@ams@flip#1{%
6992 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6993 \fi
6994 \def\bbbl@ams@preset#1{%
6995 \ifnum\bbbl@thetextdir>\z@
6996 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6997 \bbbl@sreplace\textdef@{\hbox}{\bbbl@ams@tagbox\hbox}%
6998 \bbbl@sreplace\maketag@@@{\hbox}{\bbbl@ams@tagbox#1}%
6999 \fi}%
7000 \def\bbbl@ams@equation{%
7001 \ifnum\bbbl@thetextdir>\z@
7002 \bbbl@ams@preset\hbox
7003 \chardef\bbbl@thetextdir\z@
7004 \bbbl@add\normalfont{\bbbl@eqnodir}%
7005 \ifcase\bbbl@eqnpos
7006 \def\veqno##1##2{\bbbl@eqno@flip{##1##2}}%
7007 \or
7008 \def\veqno##1##2{\bbbl@leqno@flip{##1##2}}%
7009 \fi
7010 \fi}%
7011 \AddToHook{env/equation/begin}{\bbbl@ams@equation}%
7012 \AddToHook{env/equation*/begin}{\bbbl@ams@equation}%
7013 \AddToHook{env/cases/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7014 \AddToHook{env/multline/begin}{\bbbl@ams@preset\hbox}%
7015 \AddToHook{env/gather/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7016 \AddToHook{env/gather*/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7017 \AddToHook{env/align/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7018 \AddToHook{env/align*/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7019 \AddToHook{env/alignat/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7020 \AddToHook{env/alignat*/begin}{\bbbl@ams@preset\bbbl@ams@lap}%
7021 \AddToHook{env/eqnalign/begin}{\bbbl@ams@preset\hbox}%
7022 % Hackish, for proper alignment. Don't ask me why it works!:

```

```

7023 \bbl@exp{% Avoid a 'visible' conditional
7024   \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}%
7025   \\\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
7026 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
7027 \AddToHook{env/split/before}{%
7028   \ifnum\bbl@thetextdir>\z@
7029     \bbl@ifsamestring\@currentvir{equation}%
7030     {\ifx\bbl@ams@lap\hbox % leqno
7031       \def\bbl@ams@flip#1{%
7032         \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
7033       \else
7034         \def\bbl@ams@flip#1{%
7035           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#{1}}}}%
7036         \fi}%
7037       }%
7038     \fi}%
7039 \fi\fi}
7040 \fi

Declarations specific to lua, called by \babelprovide.

7041 \def\bbl@provide@extra#1{%
7042   % == onchar ==
7043   \ifx\bbl@KVP@onchar\@nnil\else
7044     \bbl@luahyphenate
7045     \bbl@exp{%
7046       \\\AddToHook{env/document/before}{%
7047         {\let\\bbl@ifrestoring\\@firstoftwo
7048           \\\select@language{#1}{}}}%
7049       \directlua{
7050         if Babel.locale_mapped == nil then
7051           Babel.locale_mapped = true
7052           Babel.linebreaking.add_before(Babel.locale_map, 1)
7053           Babel.loc_to_scr = {}
7054           Babel.chr_to_loc = Babel.chr_to_loc or {}
7055         end
7056         Babel.locale_props[\the\localeid].letters = false
7057       }%
7058       \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
7059       \ifin@
7060         \directlua{
7061           Babel.locale_props[\the\localeid].letters = true
7062         }%
7063       \fi
7064       \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
7065       \ifin@
7066         \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
7067           \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
7068         \fi
7069         \bbl@exp{\bbl@add\\bbl@starthyphens
7070           {\bbl@patterns@lua{\languagename}}}%
7071         \directlua{
7072           if Babel.script_blocks['\bbl@cl{sbc}'] then
7073             Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
7074             Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
7075           end
7076         }%
7077       \fi
7078       \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
7079       \ifin@
7080         \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
7081         \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%
7082         \directlua{
7083           if Babel.script_blocks['\bbl@cl{sbc}'] then

```

```

7084         Babel.loc_to_scr[\the\localeid] =
7085         Babel.script_blocks['\bbl@cl{sbcp}']
7086     end}%
7087     \ifx\bbl@mapselect\undefined
7088     \AtBeginDocument{%
7089         \bbl@patchfont{\bbl@mapselect}}%
7090         {\selectfont}}%
7091     \def\bbl@mapselect{%
7092         \let\bbl@mapselect\relax
7093         \edef\bbl@prefontid{\fontid\font}}%
7094     \def\bbl@mapdir##1{%
7095         \begingroup
7096         \setbox\z@\hbox{% Force text mode
7097             \def\language{##1}%
7098             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
7099             \bbl@switchfont
7100             \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
7101                 \directlua{
7102                     Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
7103                     ['/\bbl@prefontid'] = \fontid\font\space}%
7104                 \fi}%
7105             \endgroup}%
7106         \fi
7107         \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7108     \fi
7109     \fi
7110     % == mapfont ==
7111     % For bidi texts, to switch the font based on direction. Deprecated
7112     \ifx\bbl@KVP@mapfont\@nnil\else
7113         \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{%
7114             {\bbl@error{unknown-mapfont}}}%
7115         \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}{%
7116             \bbl@ifunset{\bbl@wdir\language}{\bbl@provide@dirs{\language}}}%
7117     \ifx\bbl@mapselect\undefined
7118     \AtBeginDocument{%
7119         \bbl@patchfont{\bbl@mapselect}}%
7120         {\selectfont}}%
7121     \def\bbl@mapselect{%
7122         \let\bbl@mapselect\relax
7123         \edef\bbl@prefontid{\fontid\font}}%
7124     \def\bbl@mapdir##1{%
7125         {\def\language{##1}%
7126             \let\bbl@ifrestoring\@firstoftwo % avoid font warning
7127             \bbl@switchfont
7128             \directlua{Babel.fontmap
7129                 [\the\csname bbl@wdir@##1\endcsname]%
7130                 [\bbl@prefontid]=\fontid\font}}}%
7131     \fi
7132     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7133     \fi
7134     % == Line breaking: CJK quotes ==
7135     \ifcase\bbl@engine\or
7136         \bbl@xin{/c}{\bbl@cl{lnbrk}}%
7137     \ifin@
7138         \bbl@ifunset{\bbl@quote\language}}{%
7139             {\directlua{
7140                 Babel.locale_props[\the\localeid].cjk_quotes = {}
7141                 local cs = 'op'
7142                 for c in string.utfvalues(
7143                     [[\csname bbl@quote\language\endcsname]]) do
7144                     if Babel.cjk_characters[c].c == 'qu' then
7145                         Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
7146                     end

```

```

7147         cs = ( cs == 'op') and 'cl' or 'op'
7148     end
7149 }}%
7150 \fi
7151 \fi
7152 % == Counters: mapdigits ==
7153 % Native digits
7154 \ifx\bbk@KVP@mapdigits\@nnil\else
7155     \bbl@ifunset{bbl@dgnat@language\name}{}%
7156     {\bbl@activate@preotf
7157     \directlua{
7158         Babel.digits_mapped = true
7159         Babel.digits = Babel.digits or {}
7160         Babel.digits[\the\localeid] =
7161             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
7162         if not Babel.numbers then
7163             function Babel.numbers(head)
7164                 local LOCALE = Babel.attr_locale
7165                 local GLYPH = node.id'glyph'
7166                 local inmath = false
7167                 for item in node.traverse(head) do
7168                     if not inmath and item.id == GLYPH then
7169                         local temp = node.get_attribute(item, LOCALE)
7170                         if Babel.digits[temp] then
7171                             local chr = item.char
7172                             if chr > 47 and chr < 58 then
7173                                 item.char = Babel.digits[temp][chr-47]
7174                             end
7175                         end
7176                     elseif item.id == node.id'math' then
7177                         inmath = (item.subtype == 0)
7178                     end
7179                 end
7180                 return head
7181             end
7182         end
7183     }}%
7184 \fi
7185 % == transforms ==
7186 \ifx\bbk@KVP@transforms\@nnil\else
7187     \def\bbl@elt##1##2##3{%
7188         \in@{${transforms.}}{##1}%
7189         \ifin@
7190             \def\bbl@tempa{##1}%
7191             \bbl@replace\bbl@tempa{transforms.}{}%
7192             \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
7193         \fi}%
7194     \bbl@exp{%
7195         \\bbl@ifblank{\bbl@cl{dgnat}}%
7196         {\let\\bbl@tempa\relax}%
7197         {\def\\bbl@tempa{%
7198             \\bbl@elt{transforms.prehyphenation}%
7199             {digits.native.1.0}{([0-9])}%
7200             \\bbl@elt{transforms.prehyphenation}%
7201             {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}%
7202     \ifx\bbl@tempa\relax\else
7203         \toks@{\expandafter\expandafter\expandafter{%
7204             \csname bbl@inidata@language\endcsname}%
7205             \bbl@csarg\edef{inidata@language}%
7206             \unexpanded\expandafter{\bbl@tempa}%
7207             \the\toks@}%
7208     \fi
7209     \csname bbl@inidata@language\endcsname

```

```

7210 \bbl@release@transforms\relax % \relax closes the last item.
7211 \fi}

```

Start tabular here:

```

7212 \def\localerestoredirs{%
7213 \ifcase\bbl@thetextdir
7214 \ifnum\textdirection=\z@\else\textdirection=\z@\fi
7215 \else
7216 \ifnum\textdirection=\@ne\else\textdirection=\@ne\fi
7217 \fi
7218 \ifcase\bbl@thepardir
7219 \ifnum\pardirection=\z@\else\pardirection=\z@\bodydirection=\z@\fi
7220 \else
7221 \ifnum\pardirection=\@ne\else\pardirection=\@ne\bodydirection=\@ne\fi
7222 \fi}
7223 %
7224 \IfBabelLayout{tabular}%
7225 {\chardef\bbl@tabular@mode\z@}% All RTL
7226 {\IfBabelLayout{lrtabular}%
7227 {\chardef\bbl@tabular@mode\@ne}%
7228 {\chardef\bbl@tabular@mode\z@}% Mixed, with LTR cols
7229 %
7230 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
7231 % Redefine: vrules mess up dirs (why?).
7232 \AtBeginDocument{\def\@arstrut{\relax\copy\@arstrutbox}}%
7233 \ifcase\bbl@tabular@mode\else % 1 = Mixed
7234 \let\bbl@parabefore\relax
7235 \AddToHook{para/before}{\bbl@parabefore}
7236 \AtBeginDocument{%
7237 \bbl@replace\@tabular{\hbox\bgroup}{\hbox bdir\z@\bgroup}%
7238 \ifnum\bbl@tabular@mode=\@ne
7239 \bbl@ifunset{\@tabclassz}{\}%
7240 \bbl@exp{\% Hide conditionals
7241 \\\bbl@sreplace\\\@tabclassz
7242 {\<ifcase>\\\@chnum}%
7243 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
7244 \@ifpackageloaded{colortbl}%
7245 {\bbl@sreplace\@classz
7246 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
7247 {\@ifpackageloaded{array}%
7248 {\bbl@exp{\% Hide conditionals
7249 \\\bbl@sreplace\\\@classz
7250 {\<ifcase>\\\@chnum}%
7251 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
7252 \\\bbl@sreplace\\\@classz
7253 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
7254 {}}%
7255 \fi}%
7256 \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.

```

7257 \AtBeginDocument{%
7258 \@ifpackageloaded{multicol}%
7259 {\toks\expandafter{\multi@column@out}%
7260 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
7261 {}%
7262 \@ifpackageloaded{paracol}%
7263 {\edef\pcol@output{%
7264 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7265 {}}%
7266 \fi

```

Finish here if there in no layout.

```
7267 \ifx\bbl@opt@layout\@nnil\endinput\fi
```

\hangindent does not honour direction changes by default, so we need to redefine \@hangfrom.

```
7268 \chardef\bbl@trace@vboxdir\z@
7269 \ifnum\bbl@bidimode>\z@ % Any bidi=
7270 \IfBabelLayout{nopars}{}
7271 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7272 \IfBabelLayout{pars}
7273 {\chardef\bbl@trace@vboxdir\@ne
7274 \def\@hangfrom#1{%
7275 \setbox\@tempboxa\hbox{#1}}%
7276 \hangindent\wd\@tempboxa
7277 \ifnum\bbl@vbox@bodydir=\pardirection\else
7278 \shapemode\@ne
7279 \fi
7280 \noindent\box\@tempboxa}}
7281 {}
7282 \fi
```

We need to patch lists in documents with both LTR and RTL paragraphs. See issue #395 in GitHub. There was a partial solution, but a better one has been devised by Udi Fogiel (in 26.4).

```
7283 \IfBabelLayout{lists}
7284 {\chardef\bbl@trace@vboxdir\@ne
7285 \let\bbl@OL@list\list
7286 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
7287 \let\bbl@NL@list\list
7288 \def\bbl@listparshape#1#2#3{%
7289 \parshape #1 #2 #3 %
7290 \ifnum\bbl@vbox@bodydir=\pardirection\else
7291 \shapemode\tw@
7292 \fi}}
7293 {}
7294 %
7295 \ifcase\bbl@trace@vboxdir\else
7296 \AtBeginDocument{\chardef\bbl@vbox@bodydir\pagedirection}%
7297 \def\bbl@vbox@lists{\chardef\bbl@vbox@bodydir\bodydirection}%
7298 \let\bbl@bidi@vbox\everyvbox
7299 \@nameuse{newtoks}\everyvbox % \outer in Plain
7300 \everyvbox\expandafter{\the\bbl@bidi@vbox}%
7301 \bbl@bidi@vbox{\bbl@vbox@lists\the\everyvbox}%
7302 \fi
7303 %
7304 \IfBabelLayout{graphics}
7305 {\let\bbl@pictresetdir\relax
7306 \def\bbl@pictsetdir#1{%
7307 \ifcase\bbl@thetextdir
7308 \let\bbl@pictresetdir\relax
7309 \else
7310 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7311 \or\textdir TLT
7312 \else\bodydir TLT \textdir TLT
7313 \fi
7314 % \text|par\dir required in pgf:
7315 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7316 \fi}%
7317 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
7318 \directlua{
7319 Babel.get_picture_dir = true
7320 Babel.picture_has_bidi = 0
7321 %
7322 function Babel.picture_dir (head)
7323 if not Babel.get_picture_dir then return head end
```

```

7324     if Babel.hlist_has_bidi(head) then
7325         Babel.picture_has_bidi = 1
7326     end
7327     return head
7328 end
7329 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7330     "Babel.picture_dir")
7331 }%
7332 \AddToHook{package/graphics/after}{%
7333     \bbl@exp{%
7334         \\bbl@sreplace\\rotatebox{\hbox{{\string#2}}}
7335         {\hbox bdir\z@{\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>{{\string#2}}}}}}
7336 \AddToHook{package/graphics/after}{%
7337     \bbl@exp{%
7338         \\bbl@sreplace\\Grot@box@std{\hbox{{\string#2}}}
7339         {\hbox bdir\z@{\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>{{\string#2}}}}%
7340         \\bbl@sreplace\\Grot@box@kv{\hbox{{\string#3}}}
7341         {{\hbox bdir\z@}{\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>{{\string#3}}}}}%
7342         \\bbl@sreplace\\Grot@box{\hbox}{\hbox bdir\z@}}
7343 \AtBeginDocument{%
7344     \long\def\put(#1,#2)#3{%
7345         \@killglue
7346         % Try:
7347         \ifx\bbl@pictresetdir\relax
7348             \def\bbl@tempc{0}%
7349         \else
7350             \directlua{
7351                 Babel.get_picture_dir = true
7352                 Babel.picture_has_bidi = 0
7353             }%
7354             \setbox\z@\hb@xt\z@{%
7355                 \@defaultunitsset\@tempdimc{#1}\unitlength
7356                 \kern\@tempdimc
7357                 #3\hss}%
7358             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7359             \fi
7360             % Do:
7361             \@defaultunitsset\@tempdimc{#2}\unitlength
7362             \raise\@tempdimc\hb@xt\z@{%
7363                 \@defaultunitsset\@tempdimc{#1}\unitlength
7364                 \kern\@tempdimc
7365                 {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
7366             \ignorespaces}%
7367             \MakeRobust\put}%
7368 \AtBeginDocument
7369 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
7370 \ifx\pgfpicture\undefined\else
7371     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
7372     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
7373     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
7374 \fi
7375 \ifx\tikzpicture\undefined\else
7376     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
7377     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
7378     \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
7379     \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
7380 \fi
7381 }}
7382 {}

```

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.

```

7383 \IfBabelLayout{counters*}%
7384 {\bbl@add\bbl@opt@layout{.counters.}}%
7385 \directlua{
7386     luatexbase.add_to_callback("process_output_buffer",
7387         Babel.discard_sublr , "Babel.discard_sublr") }%
7388 }{}
7389 \IfBabelLayout{counters}%
7390 {\let\bbl@latinarabic=\@arabic
7391 \let\bbl@OL@@arabic\@arabic
7392 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7393 \@ifpackagewith{babel}{bidi=default}%
7394 {\let\bbl@asciroman=\@roman
7395 \let\bbl@OL@@roman\@roman
7396 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7397 \let\bbl@asciiRoman=\@Roman
7398 \let\bbl@OL@@roman\@Roman
7399 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7400 \let\bbl@OL@labelenumii\labelenumii
7401 \def\labelenumii{}\theenumii}%
7402 \let\bbl@OL@p@enumiii\p@enumiii
7403 \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.

```

7404 \IfBabelLayout{extras}%
7405 {\bbl@ncarg\let\bbl@OL@underline{underline }%
7406 \bbl@carg\bbl@sreplace{underline }\hbox}%
7407 {\def\bbl@insidemath{0}\hbox bdir\ifcase\bbl@thetextdir\z@else\@ne\fi}%
7408 \let\bbl@OL@LaTeXe\LaTeXe
7409 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7410 \if b\expandafter\car\f@series\@nil\boldmath\fi
7411 \babelsublr{\LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}}%
7412 {}
7413 (/luatex)

```

9.13. 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 discretionary, 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.

```

7414 (*transforms)
7415 Babel.linebreaking.replacements = {}
7416 Babel.linebreaking.replacements[0] = {} -- pre
7417 Babel.linebreaking.replacements[1] = {} -- post
7418
7419 function Babel.tovalue(v)
7420     if type(v) == 'table' then
7421         return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7422     else
7423         return v
7424     end
7425 end
7426
7427 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7428

```

```

7429 function Babel.set_hboxed(head, gc)
7430   for item in node.traverse(head) do
7431     node.set_attribute(item, Babel.attr_hboxed, 1)
7432   end
7433   return head
7434 end
7435
7436 Babel.fetch_subtext = {}
7437
7438 Babel.ignore_pre_char = function(node)
7439   return (node.lang == Babel.nohyphenation)
7440 end
7441
7442 Babel.show_transforms = false
7443
7444 -- Merging both functions doesn't seem feasible, because there are too
7445 -- many differences.
7446 Babel.fetch_subtext[0] = function(head)
7447   local word_string = ''
7448   local word_nodes = {}
7449   local lang
7450   local item = head
7451   local inmath = false
7452
7453   while item do
7454     if item.id == 11 then
7455       inmath = (item.subtype == 0)
7456     end
7457
7458     if inmath then
7459       -- pass
7460     end
7461
7462     elseif item.id == 29 then
7463       local locale = node.get_attribute(item, Babel.attr_locale)
7464
7465       if lang == locale or lang == nil then
7466         lang = lang or locale
7467         if Babel.ignore_pre_char(item) then
7468           word_string = word_string .. Babel.us_char
7469         else
7470           if node.has_attribute(item, Babel.attr_hboxed) then
7471             word_string = word_string .. Babel.us_char
7472           else
7473             word_string = word_string .. unicode.utf8.char(item.char)
7474           end
7475         end
7476         word_nodes[#word_nodes+1] = item
7477       else
7478         break
7479       end
7480
7481       elseif item.id == 12 and item.subtype == 13 then
7482         if node.has_attribute(item, Babel.attr_hboxed) then
7483           word_string = word_string .. Babel.us_char
7484         else
7485           word_string = word_string .. ' '
7486         end
7487         word_nodes[#word_nodes+1] = item
7488
7489         -- Ignore leading unrecognized nodes, too.
7490         elseif word_string ~= '' then
7491           word_string = word_string .. Babel.us_char

```

```

7492     word_nodes[#word_nodes+1] = item -- Will be ignored
7493 end
7494
7495     item = item.next
7496 end
7497
7498 -- Here and above we remove some trailing chars but not the
7499 -- corresponding nodes. But they aren't accessed.
7500 if word_string:sub(-1) == ' ' then
7501     word_string = word_string:sub(1,-2)
7502 end
7503 if Babel.show_transforms then texio.write_nl(word_string) end
7504 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7505 return word_string, word_nodes, item, lang
7506 end
7507
7508 Babel.fetch_subtext[1] = function(head)
7509     local word_string = ''
7510     local word_nodes = {}
7511     local lang
7512     local item = head
7513     local inmath = false
7514
7515     while item do
7516
7517         if item.id == 11 then
7518             inmath = (item.subtype == 0)
7519         end
7520
7521         if inmath then
7522             -- pass
7523
7524         elseif item.id == 29 then
7525             if item.lang == lang or lang == nil then
7526                 lang = lang or item.lang
7527                 if node.has_attribute(item, Babel.attr_hboxed) then
7528                     word_string = word_string .. Babel.us_char
7529                 elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7530                     word_string = word_string .. Babel.us_char
7531                 else
7532                     word_string = word_string .. unicode.utf8.char(item.char)
7533                 end
7534                 word_nodes[#word_nodes+1] = item
7535             else
7536                 break
7537             end
7538
7539         elseif item.id == 7 and item.subtype == 2 then
7540             if node.has_attribute(item, Babel.attr_hboxed) then
7541                 word_string = word_string .. Babel.us_char
7542             else
7543                 word_string = word_string .. '='
7544             end
7545             word_nodes[#word_nodes+1] = item
7546
7547         elseif item.id == 7 and item.subtype == 3 then
7548             if node.has_attribute(item, Babel.attr_hboxed) then
7549                 word_string = word_string .. Babel.us_char
7550             else
7551                 word_string = word_string .. '|'
7552             end
7553             word_nodes[#word_nodes+1] = item
7554

```

```

7555 -- (1) Go to next word if nothing was found, and (2) implicitly
7556 -- remove leading USs.
7557 elseif word_string == '' then
7558     -- pass
7559
7560 -- This is the responsible for splitting by words.
7561 elseif (item.id == 12 and item.subtype == 13) then
7562     break
7563
7564 else
7565     word_string = word_string .. Babel.us_char
7566     word_nodes[#word_nodes+1] = item -- Will be ignored
7567 end
7568
7569 item = item.next
7570 end
7571 if Babel.show_transforms then texio.write_nl(word_string) end
7572 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7573 return word_string, word_nodes, item, lang
7574 end
7575
7576 -- LuaTeX keeps its own pointer to the last node of the list passed to
7577 -- the hyphenate callback and uses it afterwards (line_break and
7578 -- filtered_hpack), so a transform freeing that node makes TeX use
7579 -- freed memory. Before transforming, the last node is replaced with a
7580 -- copy, and the original is kept out of the list, alive and with no
7581 -- links, and freed at the next call (new_ligkern walks from such a
7582 -- node to the actual end of the list).
7583
7584 do
7585     local stale_tail
7586
7587     function Babel.protect_tail(head, tail)
7588         if stale_tail then
7589             node.free(stale_tail)
7590             stale_tail = nil
7591         end
7592         if not tail or tail == head then return end
7593         local prev = tail.prev
7594         if not prev or prev.next ~= tail then
7595             prev = head
7596             while prev.next ~= tail do prev = prev.next end
7597         end
7598         local copy = node.copy(tail)
7599         prev.next = copy
7600         copy.prev = prev
7601         copy.next = nil
7602         tail.prev = nil
7603         tail.next = nil
7604         stale_tail = tail
7605     end
7606 end
7607
7608 function Babel.pre_hyphenate_replace(head)
7609     Babel.hyphenate_replace(head, 0)
7610 end
7611
7612 function Babel.post_hyphenate_replace(head)
7613     Babel.hyphenate_replace(head, 1)
7614 end
7615
7616 Babel.us_char = string.char(31)
7617

```

```

7618 function Babel.hyphenate_replace(head, mode)
7619   local u = unicode.utf8
7620   local lbkr = Babel.linebreaking.replacements[mode]
7621   local tovalue = Babel.tovalue
7622
7623   local word_head = head
7624
7625   -- Removed nodes are freed at the end of each match, because 'data'
7626   -- may still refer to them. Keyed by the direct node, so none is
7627   -- freed twice.
7628   local removed = {}
7629   local function remove_node(n)
7630     if n then
7631       node.remove(head, n)
7632       removed[node.direct.todirect(n)] = true
7633     end
7634   end
7635
7636   if Babel.show_transforms then
7637     texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7638   end
7639
7640   while true do -- for each subtext block
7641
7642     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7643
7644     if Babel.debug then
7645       print()
7646       print((mode == 0) and '@@@<' or '@@@>', w)
7647     end
7648
7649     if nw == nil and w == '' then break end
7650
7651     if not lang then goto next end
7652     if not lbkr[lang] then goto next end
7653
7654     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7655     -- loops are nested.
7656     for k=1, #lbkr[lang] do
7657       local p = lbkr[lang][k].pattern
7658       local r = lbkr[lang][k].replace
7659       local attr = lbkr[lang][k].attr or -1
7660
7661       if Babel.debug then
7662         print('*****', p, mode)
7663       end
7664
7665       -- This variable is set in some cases below to the first *byte*
7666       -- after the match, either as found by u.match (faster) or the
7667       -- computed position based on sc if w has changed.
7668       local last_match = 0
7669       local step = 0
7670
7671       -- For every match.
7672       while true do
7673         if Babel.debug then
7674           print('====')
7675         end
7676         local new -- used when inserting and removing nodes
7677         local dummy_node -- used by after
7678
7679         local matches = { u.match(w, p, last_match) }
7680

```

```

7681     if #matches < 2 then break end
7682
7683     -- Get and remove empty captures (with ())'s, which return a
7684     -- number with the position), and keep actual captures
7685     -- (from (...)), if any, in matches.
7686     local first = table.remove(matches, 1)
7687     local last = table.remove(matches, #matches)
7688     -- Non re-fetched substrings may contain \31, which separates
7689     -- subsubstrings.
7690     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7691
7692     local save_last = last -- with A()BC()D, points to D
7693
7694     -- Fix offsets, from bytes to unicode. Explained above.
7695     first = u.len(w:sub(1, first-1)) + 1
7696     last = u.len(w:sub(1, last-1)) -- now last points to C
7697
7698     -- This loop stores in a small table the nodes
7699     -- corresponding to the pattern. Used by 'data' to provide a
7700     -- predictable behavior with 'insert' (w_nodes is modified on
7701     -- the fly), and also access to 'remove'd nodes.
7702     local sc = first-1 -- Used below, too
7703     local data_nodes = {}
7704
7705     local enabled = true
7706     for q = 1, last-first+1 do
7707         data_nodes[q] = w_nodes[sc+q]
7708         if enabled
7709             and attr > -1
7710             and not node.has_attribute(data_nodes[q], attr)
7711         then
7712             enabled = false
7713         end
7714     end
7715
7716     -- This loop traverses the matched substring and takes the
7717     -- corresponding action stored in the replacement list.
7718     -- sc = the position in substr nodes / string
7719     -- rc = the replacement table index
7720     local rc = 0
7721
7722     ----- TODO. dummy_node?
7723     while rc < last-first+1 or dummy_node do -- for each replacement
7724         if Babel.debug then
7725             print('.....', rc + 1)
7726         end
7727         sc = sc + 1
7728         rc = rc + 1
7729
7730         if Babel.debug then
7731             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7732             local ss = ''
7733             for itt in node.traverse(head) do
7734                 if itt.id == 29 then
7735                     ss = ss .. unicode.utf8.char(itt.char)
7736                 else
7737                     ss = ss .. '{' .. itt.id .. '}'
7738                 end
7739             end
7740             print('*****', ss)
7741         end
7742     end
7743

```

```

7744     local crep = r[rc]
7745     local item = w_nodes[sc]
7746     local item_base = item
7747     local placeholder = Babel.us_char
7748     local d
7749
7750     if crep and crep.data then
7751         item_base = data_nodes[crep.data]
7752     end
7753
7754     if crep then
7755         step = crep.step or step
7756     end
7757
7758     if crep and crep.after then
7759         crep.insert = true
7760         if dummy_node then
7761             item = dummy_node
7762         else -- TODO. if there is a node after?
7763             d = node.copy(item_base)
7764             head, item = node.insert_after(head, item, d)
7765             dummy_node = item
7766         end
7767     end
7768
7769     if crep and not crep.after and dummy_node then
7770         remove_node(dummy_node)
7771         dummy_node = nil
7772     end
7773
7774     if not enabled then
7775         last_match = save_last
7776         goto next
7777
7778     elseif crep and next(crep) == nil then -- = {}
7779         if step == 0 then
7780             last_match = save_last    -- Optimization
7781         else
7782             last_match = utf8.offset(w, sc+step)
7783         end
7784         goto next
7785
7786     elseif crep == nil or crep.remove then
7787         remove_node(item)
7788         table.remove(w_nodes, sc)
7789         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7790         sc = sc - 1 -- Nothing has been inserted.
7791         last_match = utf8.offset(w, sc+1+step)
7792         goto next
7793
7794     elseif crep and crep.kashida then
7795         node.set_attribute(item,
7796             Babel.attr_kashida,
7797             crep.kashida)
7798         last_match = utf8.offset(w, sc+1+step)
7799         goto next
7800
7801     elseif crep and crep.string then
7802         local str = crep.string(matches)
7803         if str == '' then -- Gather with nil
7804             remove_node(item)
7805             table.remove(w_nodes, sc)
7806             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)

```

```

7807         sc = sc - 1 -- Nothing has been inserted.
7808     else
7809         local loop_first = true
7810         for s in string.utfvalues(str) do
7811             d = node.copy(item_base)
7812             d.char = s
7813             if loop_first then
7814                 loop_first = false
7815                 head, new = node.insert_before(head, item, d)
7816                 if sc == 1 then
7817                     word_head = head
7818                 end
7819                 w_nodes[sc] = d
7820                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7821             else
7822                 sc = sc + 1
7823                 head, new = node.insert_before(head, item, d)
7824                 table.insert(w_nodes, sc, new)
7825                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7826             end
7827             if Babel.debug then
7828                 print('.....', 'str')
7829                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7830             end
7831         end -- for
7832         remove_node(item)
7833     end -- if ''
7834     last_match = utf8.offset(w, sc+1+step)
7835     goto next
7836
7837 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7838     d = node.new(7, 3) -- (disc, regular)
7839     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7840     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7841     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7842     d.attr = item_base.attr
7843     if crep.pre == nil then -- TeXbook p96
7844         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7845     else
7846         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7847     end
7848     placeholder = '|'
7849     head, new = node.insert_before(head, item, d)
7850
7851 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7852     -- ERROR
7853
7854 elseif crep and crep.penalty then
7855     d = node.new(14, 0) -- (penalty, userpenalty)
7856     d.attr = item_base.attr
7857     d.penalty = tovalue(crep.penalty)
7858     head, new = node.insert_before(head, item, d)
7859
7860 elseif crep and crep.space then
7861     -- 655360 = 10 pt = 10 * 65536 sp
7862     d = node.new(12, 13) -- (glue, spaceskip)
7863     local quad = font.getfont(item_base.font).size or 655360
7864     node.setglue(d, tovalue(crep.space[1]) * quad,
7865                 tovalue(crep.space[2]) * quad,
7866                 tovalue(crep.space[3]) * quad)
7867     if mode == 0 then
7868         placeholder = ' '
7869     end

```

```

7870         head, new = node.insert_before(head, item, d)
7871
7872     elseif crep and crep.norule then
7873         -- 655360 = 10 pt = 10 * 65536 sp
7874         d = node.new(2, 3)      -- (rule, empty) = \no*rule
7875         local quad = font.getfont(item_base.font).size or 655360
7876         d.width  = tovalue(crep.norule[1]) * quad
7877         d.height = tovalue(crep.norule[2]) * quad
7878         d.depth  = tovalue(crep.norule[3]) * quad
7879         head, new = node.insert_before(head, item, d)
7880
7881     elseif crep and crep.spacefactor then
7882         d = node.new(12, 13)    -- (glue, spaceskip)
7883         local base_font = font.getfont(item_base.font)
7884         node.setglue(d,
7885             tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7886             tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7887             tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7888         if mode == 0 then
7889             placeholder = ' '
7890         end
7891         head, new = node.insert_before(head, item, d)
7892
7893     elseif mode == 0 and crep and crep.space then
7894         -- ERROR
7895
7896     elseif crep and crep.kern then
7897         d = node.new(13, 1)     -- (kern, user)
7898         local quad = font.getfont(item_base.font).size or 655360
7899         d.attr = item_base.attr
7900         d.kern = tovalue(crep.kern) * quad
7901         head, new = node.insert_before(head, item, d)
7902
7903     elseif crep and crep.node then
7904         d = node.new(crep.node[1], crep.node[2])
7905         d.attr = item_base.attr
7906         head, new = node.insert_before(head, item, d)
7907
7908     end -- i.e., replacement cases
7909
7910     -- Shared by disc, space(factor), kern, node and penalty.
7911     if sc == 1 then
7912         word_head = head
7913     end
7914     if crep.insert then
7915         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7916         table.insert(w_nodes, sc, new)
7917         last = last + 1
7918     else
7919         w_nodes[sc] = d
7920         remove_node(item)
7921         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7922     end
7923
7924     last_match = utf8.offset(w, sc+1+step)
7925
7926     ::next::
7927
7928 end -- for each replacement
7929
7930 if Babel.show_transforms then texio.write_nl('> ' .. w) end
7931 if Babel.debug then
7932     print('.....', '/')

```

```

7933         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7934     end
7935
7936     if dummy_node then
7937         remove_node(dummy_node)
7938         dummy_node = nil
7939     end
7940
7941     for n in pairs(removed) do
7942         node.direct.free(n)
7943     end
7944     removed = {}
7945
7946     end -- for match
7947
7948     end -- for patterns
7949
7950     ::next::
7951     word_head = nw
7952 end -- for substring
7953
7954 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7955 return head
7956 end
7957
7958 -- This table stores capture maps, numbered consecutively
7959 Babel.capture_maps = {}
7960
7961 function Babel.esc_hex_to_char(h)
7962     if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7963         tex.getcatcode(tonumber(h, 16)) ~= 12 then
7964         return string.format([[\\Uchar"%X ]], tonumber(h,16))
7965     else
7966         return unicode.utf8.char(tonumber(h, 16))
7967     end
7968 end
7969
7970 -- The following functions belong to the next macro
7971 function Babel.capture_func(key, cap)
7972     local ret = "[[" .. cap:gsub('{{([0-9])}}', "")..m[%1]..["] .. "]"
7973     local cnt
7974     local u = unicode.utf8
7975     ret = u.gsub(ret, '{(%x%x%x%x+)}', '\\x01%1\\x04')
7976     ret, cnt = ret:gsub('{{([0-9])|([^\^]|+)|(.-)}}', Babel.capture_func_map)
7977     ret = u.gsub(ret, '\\x01(%x%x%x%x+)%x04', Babel.esc_hex_to_char)
7978     ret = ret:gsub("%[%[%]%]%.%", '')
7979     ret = ret:gsub("%.%[%[%]%]", '')
7980     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7981 end
7982
7983 function Babel.capt_map(from, mapno)
7984     return Babel.capture_maps[mapno][from] or from
7985 end
7986
7987 -- Handle the {n|abc|ABC} syntax in captures
7988 function Babel.capture_func_map(capno, from, to)
7989     local u = unicode.utf8
7990     from = u.gsub(from, '\\x01(%x%x%x%x+)%x04',
7991         function (n)
7992             return u.char(tonumber(n, 16))
7993         end)
7994     to = u.gsub(to, '\\x01(%x%x%x%x+)%x04',
7995         function (n)

```

```

7996         return u.char(tonumber(n, 16))
7997     end)
7998     local froms = {}
7999     for s in string.utfcharacters(from) do
8000         table.insert(froms, s)
8001     end
8002     local cnt = 1
8003     table.insert(Babel.capture_maps, {})
8004     local mlen = table.getn(Babel.capture_maps)
8005     for s in string.utfcharacters(to) do
8006         Babel.capture_maps[mlen][froms[cnt]] = s
8007         cnt = cnt + 1
8008     end
8009     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
8010         (mlen) .. ").." .. "["
8011 end
8012
8013 -- Create/Extend reversed sorted list of kashida weights:
8014 function Babel.capture_kashida(key, wt)
8015     wt = tonumber(wt)
8016     if Babel.kashida_wts then
8017         for p, q in ipairs(Babel.kashida_wts) do
8018             if wt == q then
8019                 break
8020             elseif wt > q then
8021                 table.insert(Babel.kashida_wts, p, wt)
8022                 break
8023             elseif table.getn(Babel.kashida_wts) == p then
8024                 table.insert(Babel.kashida_wts, wt)
8025             end
8026         end
8027     else
8028         Babel.kashida_wts = { wt }
8029     end
8030     return 'kashida = ' .. wt
8031 end
8032
8033 function Babel.capture_node(id, subtype)
8034     local sbt = 0
8035     for k, v in pairs(node.subtypes(id)) do
8036         if v == subtype then sbt = k end
8037     end
8038     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
8039 end
8040
8041 -- Experimental: applies prehyphenation transforms to a string (letters
8042 -- and spaces).
8043 function Babel.string_prehyphenation(str, locale)
8044     local n, head, last, res
8045     head = node.new(8, 0) -- dummy (hack just to start)
8046     last = head
8047     for s in string.utfvalues(str) do
8048         if s == 20 then
8049             n = node.new(12, 0)
8050         else
8051             n = node.new(29, 0)
8052             n.char = s
8053         end
8054         node.set_attribute(n, Babel.attr_locale, locale)
8055         last.next = n
8056         last = n
8057     end
8058     head = Babel.hyphenate_replace(head, 0)

```

```

8059 res = ''
8060 for n in node.traverse(head) do
8061   if n.id == 12 then
8062     res = res .. ' '
8063   elseif n.id == 29 then
8064     res = res .. unicode.utf8.char(n.char)
8065   end
8066 end
8067 node.flush_list(head)
8068 tex.print(res)
8069 end
8070 (/transforms)

```

9.14. 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):

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

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

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

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

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

```

8071 (*basic-r)
8072 Babel.bidi_enabled = true
8073
8074 require('babel-data-bidi.lua')
8075
8076 local characters = Babel.characters
8077 local ranges = Babel.ranges
8078
8079 local DIR = node.id("dir")
8080
8081 local function dir_mark(head, from, to, outer)
8082   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse

```

```

8083 local d = node.new(DIR)
8084 d.dir = '+' .. dir
8085 node.insert_before(head, from, d)
8086 d = node.new(DIR)
8087 d.dir = '-' .. dir
8088 node.insert_after(head, to, d)
8089 end
8090
8091 function Babel.bidi(head, ispar)
8092   local first_n, last_n      -- first and last char with nums
8093   local last_es              -- an auxiliary 'last' used with nums
8094   local first_d, last_d      -- first and last char in L/R block
8095   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):

```

8096 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
8097 local strong_lr = (strong == 'l') and 'l' or 'r'
8098 local outer = strong
8099
8100 local new_dir = false
8101 local first_dir = false
8102 local inmath = false
8103
8104 local last_lr
8105
8106 local type_n = ''
8107
8108 for item in node.traverse(head) do
8109   -- three cases: glyph, dir, otherwise
8110   if item.id == node.id'glyph'
8111     or (item.id == 7 and item.subtype == 2) then
8112     local itemchar
8113     if item.id == 7 and item.subtype == 2 then
8114       itemchar = item.replace.char
8115     else
8116       itemchar = item.char
8117     end
8118     local chardata = characters[itemchar]
8119     dir = chardata and chardata.d or nil
8120     if not dir then
8121       for nn, et in ipairs(ranges) do
8122         if itemchar < et[1] then
8123           break
8124         elseif itemchar <= et[2] then
8125           dir = et[3]
8126           break
8127         end
8128       end
8129     end
8130     dir = dir or 'l'
8131     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).

```

8134   if new_dir then
8135     attr_dir = 0
8136     for at in node.traverse(item.attr) do

```

```

8137         if at.number == Babel.attr_dir then
8138             attr_dir = at.value & 0x3
8139         end
8140     end
8141     if attr_dir == 1 then
8142         strong = 'r'
8143     elseif attr_dir == 2 then
8144         strong = 'al'
8145     else
8146         strong = 'l'
8147     end
8148     strong_lr = (strong == 'l') and 'l' or 'r'
8149     outer = strong_lr
8150     new_dir = false
8151 end
8152
8153 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

8154     dir_real = dir          -- We need dir_real to set strong below
8155     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:

```

8156     if strong == 'al' then
8157         if dir == 'en' then dir = 'an' end          -- W2
8158         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
8159         strong_lr = 'r'                             -- W3
8160     end

```

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

```

8161     elseif item.id == node.id'dir' and not inmath then
8162         new_dir = true
8163         dir = nil
8164     elseif item.id == node.id'math' then
8165         inmath = (item.subtype == 0)
8166     else
8167         dir = nil          -- Not a char
8168     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 `texdir` 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>`.

```

8169     if dir == 'en' or dir == 'an' or dir == 'et' then
8170         if dir ~= 'et' then
8171             type_n = dir
8172         end
8173         first_n = first_n or item
8174         last_n = last_es or item
8175         last_es = nil
8176     elseif dir == 'es' and last_n then -- W3+W6
8177         last_es = item
8178     elseif dir == 'cs' then          -- it's right - do nothing
8179     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
8180         if strong_lr == 'r' and type_n ~= '' then
8181             dir_mark(head, first_n, last_n, 'r')
8182         elseif strong_lr == 'l' and first_d and type_n == 'an' then
8183             dir_mark(head, first_n, last_n, 'r')
8184             dir_mark(head, first_d, last_d, outer)
8185             first_d, last_d = nil, nil
8186         elseif strong_lr == 'l' and type_n ~= '' then
8187             last_d = last_n

```

```

8188     end
8189     type_n = ''
8190     first_n, last_n = nil, nil
8191 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:

```

8192 if dir == 'l' or dir == 'r' then
8193   if dir ~= outer then
8194     first_d = first_d or item
8195     last_d = item
8196   elseif first_d and dir ~= strong_lr then
8197     dir_mark(head, first_d, last_d, outer)
8198     first_d, last_d = nil, nil
8199   end
8200 end

```

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

```

8201 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
8202   item.char = characters[item.char] and
8203     characters[item.char].m or item.char
8204 elseif (dir or new_dir) and last_lr ~= item then
8205   local mir = outer .. strong_lr .. (dir or outer)
8206   if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
8207     for ch in node.traverse(node.next(last_lr)) do
8208       if ch == item then break end
8209       if ch.id == node.id'glyph' and characters[ch.char] then
8210         ch.char = characters[ch.char].m or ch.char
8211       end
8212     end
8213   end
8214 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).

```

8215 if dir == 'l' or dir == 'r' then
8216   last_lr = item
8217   strong = dir_real -- Don't search back - best save now
8218   strong_lr = (strong == 'l') and 'l' or 'r'
8219 elseif new_dir then
8220   last_lr = nil
8221 end
8222 end

```

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

```

8223 if last_lr and outer == 'r' then
8224   for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
8225     if characters[ch.char] then
8226       ch.char = characters[ch.char].m or ch.char
8227     end
8228   end
8229 end
8230 if first_n then
8231   dir_mark(head, first_n, last_n, outer)
8232 end
8233 if first_d then
8234   dir_mark(head, first_d, last_d, outer)
8235 end

```

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

```
8236 return node.prev(head) or head
8237 end
8238 (/basic-r)
```

And here the Lua code for bidi=basic:

```
8239 (*basic)
8240 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
8241
8242 Babel.fontmap = Babel.fontmap or {}
8243 Babel.fontmap[0] = {}      -- l
8244 Babel.fontmap[1] = {}      -- r
8245 Babel.fontmap[2] = {}      -- al/an
8246
8247 -- To cancel mirroring. Also OML, OMS, U?
8248 Babel.symbol_fonts = Babel.symbol_fonts or {}
8249 Babel.symbol_fonts[font.id('tenln')] = true
8250 Babel.symbol_fonts[font.id('tenlnw')] = true
8251 Babel.symbol_fonts[font.id('tencirc')] = true
8252 Babel.symbol_fonts[font.id('tencircw')] = true
8253
8254 Babel.bidi_enabled = true
8255 Babel.mirroring_enabled = true
8256
8257 require('babel-data-bidi.lua')
8258
8259 local characters = Babel.characters
8260 local ranges = Babel.ranges
8261
8262 local DIR = node.id('dir')
8263 local GLYPH = node.id('glyph')
8264
8265 local function insert_implicit(head, state, outer)
8266   local new_state = state
8267   if state.sim and state.eim and state.sim ~= state.eim then
8268     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
8269     local d = node.new(DIR)
8270     d.dir = '+' .. dir
8271     node.insert_before(head, state.sim, d)
8272     local d = node.new(DIR)
8273     d.dir = '-' .. dir
8274     node.insert_after(head, state.eim, d)
8275   end
8276   new_state.sim, new_state.eim = nil, nil
8277   return head, new_state
8278 end
8279
8280 local function insert_numeric(head, state)
8281   local new
8282   local new_state = state
8283   if state.san and state.ean and state.san ~= state.ean then
8284     local d = node.new(DIR)
8285     d.dir = '+TLT'
8286     _, new = node.insert_before(head, state.san, d)
8287     if state.san == state.sim then state.sim = new end
8288     local d = node.new(DIR)
8289     d.dir = '-TLT'
8290     _, new = node.insert_after(head, state.ean, d)
8291     if state.ean == state.eim then state.eim = new end
8292   end
8293   new_state.san, new_state.ean = nil, nil
8294   return head, new_state
```

```

8295 end
8296
8297 local function glyph_not_symbol_font(node)
8298   if node.id == GLYPH then
8299     return not Babel.symbol_fonts[node.font]
8300   else
8301     return false
8302   end
8303 end
8304
8305 -- TODO - \hbox with an explicit dir can lead to wrong results
8306 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
8307 -- was made to improve the situation, but the problem is the 3-dir
8308 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8309 -- well.
8310
8311 function Babel.bidi(head, ispar, hdir)
8312   local d -- d is used mainly for computations in a loop
8313   local prev_d = ''
8314   local new_d = false
8315
8316   local nodes = {}
8317   local outer_first = nil
8318   local inmath = false
8319
8320   local glue_d = nil
8321   local glue_i = nil
8322
8323   local has_en = false
8324   local first_et = nil
8325
8326   local has_hyperlink = false
8327
8328   local ATDIR = Babel.attr_dir
8329   local attr_d, temp
8330   local locale_d
8331
8332   local save_outer
8333   local locale_d = node.get_attribute(head, ATDIR)
8334   if locale_d then
8335     locale_d = locale_d & 0x3
8336     save_outer = (locale_d == 0 and 'l') or
8337                  (locale_d == 1 and 'r') or
8338                  (locale_d == 2 and 'al')
8339   elseif ispar then -- Or error? Shouldn't happen
8340     -- when the callback is called, we are just _after_ the box,
8341     -- and the textdir is that of the surrounding text
8342     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8343   else -- Empty box
8344     save_outer = ('TRT' == hdir) and 'r' or 'l'
8345   end
8346   local outer = save_outer
8347   local last = outer
8348   -- 'al' is only taken into account in the first, current loop
8349   if save_outer == 'al' then save_outer = 'r' end
8350
8351   local fontmap = Babel.fontmap
8352
8353   for item in node.traverse(head) do
8354     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8355     locale_d = node.get_attribute(item, ATDIR)
8356     node.set_attribute(item, ATDIR, 0x80)

```

```

8358
8359 -- In what follows, #node is the last (previous) node, because the
8360 -- current one is not added until we start processing the neutrals.
8361 -- three cases: glyph, dir, otherwise
8362 if glyph_not_symbol_font(item)
8363     or (item.id == 7 and item.subtype == 2) then
8364
8365     if inmath then goto nextnode end
8366
8367     if locale_d == 0x80 then goto nextnode end
8368     locale_d = locale_d or ((save_outer=='l') and 0 or 1)
8369
8370     local d_font = nil
8371     local item_r
8372     if item.id == 7 and item.subtype == 2 then
8373         item_r = item.replace -- automatic discs have just 1 glyph
8374     else
8375         item_r = item
8376     end
8377
8378     local chardata = characters[item_r.char]
8379     d = chardata and chardata.d or nil
8380     if not d or d == 'nsm' then
8381         for nn, et in ipairs(ranges) do
8382             if item_r.char < et[1] then
8383                 break
8384             elseif item_r.char <= et[2] then
8385                 if not d then d = et[3]
8386                 elseif d == 'nsm' then d_font = et[3]
8387                 end
8388                 break
8389             end
8390         end
8391     end
8392     d = d or 'l'
8393
8394     -- A short 'pause' in bidi for mapfont
8395     -- %%% TODO. move if fontmap here
8396     d_font = d_font or d
8397     d_font = (d_font == 'l' and 0) or
8398             (d_font == 'nsm' and 0) or
8399             (d_font == 'r' and 1) or
8400             (d_font == 'al' and 2) or
8401             (d_font == 'an' and 2) or nil
8402     if d_font and fontmap and fontmap[d_font][item_r.font] then
8403         item_r.font = fontmap[d_font][item_r.font]
8404     end
8405
8406     if new_d then
8407         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8408         if inmath then
8409             attr_d = 0
8410         else
8411             attr_d = locale_d & 0x3
8412         end
8413         if attr_d == 1 then
8414             outer_first = 'r'
8415             last = 'r'
8416         elseif attr_d == 2 then
8417             outer_first = 'r'
8418             last = 'al'
8419         else
8420             outer_first = 'l'

```

```

8421         last = 'l'
8422     end
8423     outer = last
8424     has_en = false
8425     first_et = nil
8426     new_d = false
8427 end
8428
8429 if glue_d then
8430     if (d == 'l' and 'l' or 'r') ~= glue_d then
8431         table.insert(nodes, {glue_i, 'on', nil})
8432     end
8433     glue_d = nil
8434     glue_i = nil
8435 end
8436
8437 elseif item.id == DIR then
8438     if inmath then goto nextnode end
8439     d = nil
8440     new_d = true
8441
8442 elseif item.id == node.id'glue' and item.subtype == 13 then
8443     if inmath then goto nextnode end
8444     glue_d = d
8445     glue_i = item
8446     d = nil
8447
8448 elseif item.id == node.id'math' then
8449     if item.subtype == 0 then -- mathon
8450         inmath = true
8451         d = 'on'
8452         table.insert(nodes, {item, 'on', outer_first})
8453         outer_first = nil
8454         goto nextnode
8455     else -- mathoff
8456         inmath = false
8457     end
8458
8459 elseif item.id == 8 and item.subtype == 19 then
8460     has_hyperlink = true
8461
8462 else
8463     d = nil
8464 end
8465
8466 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8467 if last == 'al' and d == 'en' then
8468     d = 'an'          -- W3
8469 elseif last == 'al' and (d == 'et' or d == 'es') then
8470     d = 'on'          -- W6
8471 end
8472
8473 -- EN + CS/ES + EN      -- W4
8474 if d == 'en' and #nodes >= 2 then
8475     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8476         and nodes[#nodes-1][2] == 'en' then
8477         nodes[#nodes][2] = 'en'
8478     end
8479 end
8480
8481 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8482 if d == 'an' and #nodes >= 2 then
8483     if (nodes[#nodes][2] == 'cs')

```

```

8484         and nodes[#nodes-1][2] == 'an' then
8485             nodes[#nodes][2] = 'an'
8486         end
8487     end
8488
8489     -- ET/EN             -- W5 + W7->l / W6->on
8490     if d == 'et' then
8491         first_et = first_et or (#nodes + 1)
8492     elseif d == 'en' then
8493         has_en = true
8494         first_et = first_et or (#nodes + 1)
8495     elseif first_et then    -- d may be nil here !
8496         if has_en then
8497             if last == 'l' then
8498                 temp = 'l'    -- W7
8499             else
8500                 temp = 'en'    -- W5
8501             end
8502         else
8503             temp = 'on'        -- W6
8504         end
8505         for e = first_et, #nodes do
8506             if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8507         end
8508         first_et = nil
8509         has_en = false
8510     end
8511
8512     -- Force mathdir in math if ON (currently works as expected only
8513     -- with 'l')
8514
8515     if inmath and d == 'on' then
8516         d = ('TRT' == tex.mathdir) and 'r' or 'l'
8517     end
8518
8519     if d then
8520         if d == 'al' then
8521             d = 'r'
8522             last = 'al'
8523         elseif d == 'l' or d == 'r' then
8524             last = d
8525         end
8526         prev_d = d
8527         table.insert(nodes, {item, d, outer_first})
8528     end
8529
8530     outer_first = nil
8531
8532     ::nextnode::
8533
8534 end -- for each node
8535
8536 -- TODO -- repeated here in case EN/ET is the last node. Find a
8537 -- better way of doing things:
8538 if first_et then    -- dir may be nil here !
8539     if has_en then
8540         if last == 'l' then
8541             temp = 'l'    -- W7
8542         else
8543             temp = 'en'    -- W5
8544         end
8545     else
8546         temp = 'on'        -- W6

```

```

8547     end
8548     for e = first_et, #nodes do
8549         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8550     end
8551 end
8552
8553 -- dummy node, to close things
8554 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8555
8556 ----- NEUTRAL -----
8557
8558 outer = save_outer
8559 last = outer
8560
8561 local first_on = nil
8562
8563 for q = 1, #nodes do
8564     local item
8565
8566     local outer_first = nodes[q][3]
8567     outer = outer_first or outer
8568     last = outer_first or last
8569
8570     local d = nodes[q][2]
8571     if d == 'an' or d == 'en' then d = 'r' end
8572     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8573
8574     if d == 'on' then
8575         first_on = first_on or q
8576     elseif first_on then
8577         if last == d then
8578             temp = d
8579         else
8580             temp = outer
8581         end
8582         for r = first_on, q - 1 do
8583             nodes[r][2] = temp
8584             item = nodes[r][1] -- MIRRORING
8585             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8586                 and temp == 'r' and characters[item.char] then
8587                 local font_mode = ''
8588                 if item.font > 0 and font.fonts[item.font].properties then
8589                     font_mode = font.fonts[item.font].properties.mode
8590                 end
8591                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8592                     item.char = characters[item.char].m or item.char
8593                 end
8594             end
8595         end
8596         first_on = nil
8597     end
8598
8599     if d == 'r' or d == 'l' then last = d end
8600 end
8601
8602 ----- IMPLICIT, REORDER -----
8603
8604 outer = save_outer
8605 last = outer
8606
8607 local state = {}
8608 state.has_r = false
8609

```

```

8610 for q = 1, #nodes do
8611
8612     local item = nodes[q][1]
8613
8614     outer = nodes[q][3] or outer
8615
8616     local d = nodes[q][2]
8617
8618     if d == 'nsm' then d = last end          -- W1
8619     if d == 'en' then d = 'an' end
8620     local isdir = (d == 'r' or d == 'l')
8621
8622     if outer == 'l' and d == 'an' then
8623         state.san = state.san or item
8624         state.ean = item
8625     elseif state.san then
8626         head, state = insert_numeric(head, state)
8627     end
8628
8629     if outer == 'l' then
8630         if d == 'an' or d == 'r' then      -- im -> implicit
8631             if d == 'r' then state.has_r = true end
8632             state.sim = state.sim or item
8633             state.eim = item
8634         elseif d == 'l' and state.sim and state.has_r then
8635             head, state = insert_implicit(head, state, outer)
8636         elseif d == 'l' then
8637             state.sim, state.eim, state.has_r = nil, nil, false
8638         end
8639     else
8640         if d == 'an' or d == 'l' then
8641             if nodes[q][3] then -- nil except after an explicit dir
8642                 state.sim = item -- so we move sim 'inside' the group
8643             else
8644                 state.sim = state.sim or item
8645             end
8646             state.eim = item
8647         elseif d == 'r' and state.sim then
8648             head, state = insert_implicit(head, state, outer)
8649         elseif d == 'r' then
8650             state.sim, state.eim = nil, nil
8651         end
8652     end
8653
8654     if isdir then
8655         last = d          -- Don't search back - best save now
8656     elseif d == 'on' and state.san then
8657         state.san = state.san or item
8658         state.ean = item
8659     end
8660
8661 end
8662
8663 head = node.prev(head) or head
8664 % \end{macrocode}
8665 %
8666 % Now direction nodes has been distributed with relation to characters
8667 % and spaces, we need to take into account \TeX-specific elements in
8668 % the node list, to move them at an appropriate place. Firstly, with
8669 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8670 % that the latter are still discardable.
8671 %
8672 % \begin{macrocode}

```

```

8673 --- FIXES ---
8674 if has_hyperlink then
8675     local flag, linking = 0, 0
8676     for item in node.traverse(head) do
8677         if item.id == DIR then
8678             if item.dir == '+TRT' or item.dir == '+TLT' then
8679                 flag = flag + 1
8680             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8681                 flag = flag - 1
8682             end
8683             elseif item.id == 8 and item.subtype == 19 then
8684                 linking = flag
8685             elseif item.id == 8 and item.subtype == 20 then
8686                 if linking > 0 then
8687                     if item.prev.id == DIR and
8688                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8689                         local old = item.prev
8690                         d = node.new(DIR)
8691                         d.dir = old.dir
8692                         head = node.remove(head, old)
8693                         node.free(old)
8694                         node.insert_after(head, item, d)
8695                     end
8696                 end
8697                 linking = 0
8698             end
8699         end
8700     end
8701
8702     local item, next_item = head
8703     while item do
8704         next_item = item.next
8705         if item.id == 10 then
8706             local p = item
8707             while p.prev and p.prev.id == 14 do
8708                 p = p.prev
8709             end
8710             if p ~= item then
8711                 head = node.remove(head, item)
8712                 head = node.insert_before(head, p, item)
8713             end
8714         end
8715         item = next_item
8716     end
8717
8718     return head
8719 end
8720
8720 function Babel.unset_atdir(head)
8721     local ATDIR = Babel.attr_dir
8722     for item in node.traverse(head) do
8723         node.set_attribute(item, ATDIR, 0x80)
8724     end
8725     return head
8726 end

```

The following code has been contributed by Udi Fogiel. It fixes misplaced glues when combined with penalties at line breaks.

```

8727 local GLUE    = node.id'glue'
8728 local PENALTY = node.id'penalty'
8729 local HLIST    = node.id'hlist'
8730 local LINE     = table.swapped(node.subtypes("hlist")).line
8731
8732 local characters = Babel.characters

```

```

8733 local ranges      = Babel.ranges
8734
8735 -- L1's whitespace set: WS + isolate formats + (per the UBA reference) the
8736 -- explicit embedding/override formats + BN.
8737 -- Babel does not use embedding/override/isolates. In fact, their use
8738 -- is discouraged by the UAX#9. To be removed.
8739 local ll_whitespace = {
8740   ws = true, bn = true,
8741   lre = true, rle = true, lro = true, rlo = true, pdf = true,
8742   lri = true, rli = true, fsi = true, pdi = true,
8743 }
8744
8745 local function bidi_class(c)
8746   local cd = characters[c]
8747   local d = cd and cd.d
8748   if not d then
8749     for _, et in ipairs(ranges) do
8750       if c < et[1] then
8751         break
8752       elseif c <= et[2] then
8753         d = et[3]
8754         break
8755       end
8756     end
8757   end
8758   return d or 'l'
8759 end
8760
8761 -- Nodes we step over as part of a trailing-whitespace run: glue,
8762 -- penalties, and a whitespace glyph.
8763 local function skippable(n)
8764   local id = n.id
8765   if id == GLUE or id == PENALTY then
8766     return true
8767   elseif id == GLYPH then
8768     return ll_whitespace[bidi_class(n.char)] == true
8769   end
8770   return false
8771 end
8772
8773 local function is_enddir(n)
8774   return n.id == DIR and n.subtype == 1
8775 end
8776
8777 -- Rule L1, clause 4, for one line: pull its trailing whitespace out past the
8778 -- closing dir run so it renders in the box (base) direction.
8779 local function reset_line(head)
8780   local last = node.slide(head) -- tail, fixes prev links
8781   -- 1. skip the tail that follows the closing dir run (rightskip/parfillskip/...)
8782   local q = last
8783   while q and skippable(q) do q = q.prev end
8784   if not (q and is_enddir(q)) then return head end
8785   local d_last = q
8786   -- 2. step back over the whole closing dir run
8787   while q and q.id == DIR do q = q.prev end
8788   -- 3. what precedes the dirs must be the trailing whitespace run
8789   if not (q and skippable(q)) then return head end
8790   local w_last = q
8791   while q and skippable(q) do q = q.prev end
8792   local before_w = q
8793   if not before_w then return head end -- whole line is whitespace; leave it
8794   local w_first = before_w.next
8795   local d_first = w_last.next

```

```

8796 local after_d = d_last.next
8797 -- detach W = [w_first .. w_last]
8798 before_w.next = d_first
8799 if d_first then d_first.prev = before_w end
8800 -- reinsert W after the dir run (outside every dir => box/base direction)
8801 d_last.next = w_first
8802 w_first.prev = d_last
8803 w_last.next = after_d
8804 if after_d then after_d.prev = w_last end
8805 return head
8806 end
8807
8808 function Babel.reset_lines(head)
8809 for line, subtype in node.traverse_id(HLIST, head) do
8810   if subtype == LINE then
8811     line.list = reset_line(line.list)
8812   end
8813 end
8814 return head
8815 end
8816
8817 luatexbase.add_to_callback('post_linebreak_filter',
8818   function(head, gc)
8819     if Babel.bidi_enabled then head = Babel.reset_lines(head) end
8820     return head
8821   end,
8822   'Babel.reset_lines')
8823 (/basic)

```

10. 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.

11. 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.

```

8824 (*nil)
8825 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8826 \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.

```

8827 \ifx\l@nil\undefined
8828   \newlanguage\l@nil
8829   \namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8830   \let\bbl@elt\relax
8831   \edef\bbl@languages{% Add it to the list of languages
8832     \bbl@languages\bbl@elt{nil}{the\l@nil}}}}
8833 \fi

```

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

```
8834 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil

\datenil

```
8835 \let\captionnil\@empty
```

```
8836 \let\datenil\@empty
```

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

```
8837 \def\bbl@inidata@nil{%
8838   \bbl@elt{identification}{tag.ini}{und}%
8839   \bbl@elt{identification}{load.level}{0}%
8840   \bbl@elt{identification}{charset}{utf8}%
8841   \bbl@elt{identification}{version}{1.0}%
8842   \bbl@elt{identification}{date}{2022-05-16}%
8843   \bbl@elt{identification}{name.local}{nil}%
8844   \bbl@elt{identification}{name.english}{nil}%
8845   \bbl@elt{identification}{name.babel}{nil}%
8846   \bbl@elt{identification}{tag.bcp47}{und}%
8847   \bbl@elt{identification}{language.tag.bcp47}{und}%
8848   \bbl@elt{identification}{tag.opentype}{dflt}%
8849   \bbl@elt{identification}{script.name}{Latin}%
8850   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8851   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8852   \bbl@elt{identification}{level}{1}%
8853   \bbl@elt{identification}{encodings}{}%
8854   \bbl@elt{identification}{derivate}{no}}
8855 \@namedef{bbl@tbc@nil}{und}
8856 \@namedef{bbl@lbc@nil}{und}
8857 \@namedef{bbl@casing@nil}{und}
8858 \@namedef{bbl@lotf@nil}{dflt}
8859 \@namedef{bbl@elname@nil}{nil}
8860 \@namedef{bbl@lname@nil}{nil}
8861 \@namedef{bbl@esname@nil}{Latin}
8862 \@namedef{bbl@sname@nil}{Latin}
8863 \@namedef{bbl@sbc@nil}{Latn}
8864 \@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.

```
8865 \ldf@finish{nil}
```

```
8866 (/nil)
```

12. 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.

```
8867 (<(*Compute Julian day)) ▶
8868 \def\bbl@fpmmod#1#2{(#1-#2*floor((#1)/(#2)))}
8869 \def\bbl@cs@gregleap#1{%
8870   (\bbl@fpmmod{#1}{4} == 0) &&
8871   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8872 \def\bbl@cs@jd#1#2#3{% year, month, day
8873   \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8874     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8875     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8876     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 }}
8877 (</Compute Julian day))
```

12.1. Islamic

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

```
8878 (*ca-islamic)
8879 <@Compute Julian day@>
8880 % == islamic (default)
8881 % Not yet implemented
8882 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}
```

The Civil calendar.

```
8883 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8884 ((#3 + ceil(29.5 * (#2 - 1)) +
8885 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8886 1948439.5) - 1) }
8887 \@namedef\bbl@ca@islamic-civil++{\bbl@ca@islamicvl@x{+2}}
8888 \@namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl@x{+1}}
8889 \@namedef\bbl@ca@islamic-civil{\bbl@ca@islamicvl@x{}}
8890 \@namedef\bbl@ca@islamic-civil-{\bbl@ca@islamicvl@x{-1}}
8891 \@namedef\bbl@ca@islamic-civil--{\bbl@ca@islamicvl@x{-2}}
8892 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8893 \edef\bbl@tempa{%
8894 \fpeval{ floor(\bbl@cs@jld{#2}{#3}{#4})+0.5 #1}}%
8895 \edef#5{%
8896 \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8897 \edef#6{\fpeval{
8898 min(12,ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8899 \edef#7{\fpeval{ \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).

```
8900 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8901 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8902 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8903 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8904 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8905 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8906 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8907 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8908 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8909 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8910 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8911 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8912 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8913 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8914 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8915 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8916 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8917 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8918 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8919 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8920 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8921 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8922 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8923 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8924 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8925 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8926 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8927 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8928 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8929 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8930 65401,65431,65460,65490,65520}
8931 \@namedef\bbl@ca@islamic-umalqura+{\bbl@ca@islamcuqr@x{+1}}
```

```

8932 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8933 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8934 \def\bbl@ca@islamcuqr@x#1#2-#3-#4@@#5#6#7{%
8935   \ifnum#2>2014 \ifnum#2<2038
8936     \bbl@afterfi\expandafter\@gobble
8937   \fi\fi
8938   {\bbl@error{year-out-range}{2014-2038}}{}}%
8939 \edef\bbl@tempd{\fpeval{ % (Julian) day
8940   \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8941 \count@\@ne
8942 \bbl@foreach\bbl@cs@umalqura@data{%
8943   \advance\count@\@ne
8944   \ifnum##1>\bbl@tempd\else
8945     \edef\bbl@tempe{\the\count@}%
8946     \edef\bbl@tempb{##1}%
8947     \fi}%
8948 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8949 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1 ) / 12) }}% annus
8950 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8951 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8952 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8953 \bbl@add\bbl@precalendar{%
8954   \bbl@replace\bbl@ld@calendar{-civil}}}%
8955 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8956 \bbl@replace\bbl@ld@calendar{+}}}%
8957 \bbl@replace\bbl@ld@calendar{-}}}%
8958 (/ca-islamic)

```

12.2. 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 hebcal.sty

```

8959 (*ca-hebrew)
8960 \newcount\bbl@cntcommon
8961 \def\bbl@remainder#1#2#3{%
8962   #3=#1\relax
8963   \divide #3 by #2\relax
8964   \multiply #3 by -#2\relax
8965   \advance #3 by #1\relax}%
8966 \newif\ifbbl@divisible
8967 \def\bbl@checkifdivisible#1#2{%
8968   {\countdef\tmp=0
8969   \bbl@remainder{#1}{#2}{\tmp}%
8970   \ifnum \tmp=0
8971     \global\bbl@divisibletrue
8972   \else
8973     \global\bbl@divisiblefalse
8974   \fi}}
8975 \newif\ifbbl@gregleap
8976 \def\bbl@ifgregleap#1{%
8977   \bbl@checkifdivisible{#1}{4}%
8978   \ifbbl@divisible
8979     \bbl@checkifdivisible{#1}{100}%
8980     \ifbbl@divisible
8981       \bbl@checkifdivisible{#1}{400}%
8982       \ifbbl@divisible
8983         \bbl@gregleaptrue
8984       \else
8985         \bbl@gregleapfalse
8986       \fi
8987     \else
8988       \bbl@gregleaptrue

```

```

8989     \fi
8990 \else
8991     \bbl@gregleapfalse
8992 \fi
8993 \ifbbl@gregleap}
8994 \def\bbl@gregdayspriormonths#1#2#3{%
8995     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8996         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8997     \bbl@ifgregleap{#2}%
8998     \ifnum #1 > 2
8999         \advance #3 by 1
9000     \fi
9001 \fi
9002 \global\bbl@cntcommon=#3}%
9003 #3=\bbl@cntcommon}
9004 \def\bbl@gregdaysprioryears#1#2{%
9005     {\countdef\tmpc=4
9006     \countdef\tmpb=2
9007     \tmpb=#1\relax
9008     \advance \tmpb by -1
9009     \tmpc=\tmpb
9010     \multiply \tmpc by 365
9011     #2=\tmpc
9012     \tmpc=\tmpb
9013     \divide \tmpc by 4
9014     \advance #2 by \tmpc
9015     \tmpc=\tmpb
9016     \divide \tmpc by 100
9017     \advance #2 by -\tmpc
9018     \tmpc=\tmpb
9019     \divide \tmpc by 400
9020     \advance #2 by \tmpc
9021     \global\bbl@cntcommon=#2\relax}%
9022 #2=\bbl@cntcommon}
9023 \def\bbl@absfromgreg#1#2#3#4{%
9024     {\countdef\tmpd=0
9025     #4=#1\relax
9026     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
9027     \advance #4 by \tmpd
9028     \bbl@gregdaysprioryears{#3}{\tmpd}%
9029     \advance #4 by \tmpd
9030     \global\bbl@cntcommon=#4\relax}%
9031 #4=\bbl@cntcommon}
9032 \newif\ifbbl@hebrleap
9033 \def\bbl@checkleaphebyear#1{%
9034     {\countdef\tmpa=0
9035     \countdef\tmpb=1
9036     \tmpa=#1\relax
9037     \multiply \tmpa by 7
9038     \advance \tmpa by 1
9039     \bbl@remainder{\tmpa}{19}{\tmpb}%
9040     \ifnum \tmpb < 7
9041         \global\bbl@hebrleaptrue
9042     \else
9043         \global\bbl@hebrleapfalse
9044     \fi}}
9045 \def\bbl@hebrlapsedmonths#1#2{%
9046     {\countdef\tmpa=0
9047     \countdef\tmpb=1
9048     \countdef\tmpc=2
9049     \tmpa=#1\relax
9050     \advance \tmpa by -1
9051     #2=\tmpa

```

```

9052 \divide #2 by 19
9053 \multiply #2 by 235
9054 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
9055 \tmpc=\tmpb
9056 \multiply \tmpb by 12
9057 \advance #2 by \tmpb
9058 \multiply \tmpc by 7
9059 \advance \tmpc by 1
9060 \divide \tmpc by 19
9061 \advance #2 by \tmpc
9062 \global\bbl@cntcommon=#2}%
9063 #2=\bbl@cntcommon}
9064 \def\bbl@hebreleapseddays#1#2{%
9065 {\countdef\tmpa=0
9066 \countdef\tmpb=1
9067 \countdef\tmpc=2
9068 \bbl@hebreleapsedmonths{#1}{#2}%
9069 \tmpa=#2\relax
9070 \multiply \tmpa by 13753
9071 \advance \tmpa by 5604
9072 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
9073 \divide \tmpa by 25920
9074 \multiply #2 by 29
9075 \advance #2 by 1
9076 \advance #2 by \tmpa
9077 \bbl@remainder{#2}{7}{\tmpa}%
9078 \ifnum \tmpc < 19440
9079 \ifnum \tmpc < 9924
9080 \else
9081 \ifnum \tmpa=2
9082 \bbl@checkleaphebrewyear{#1}% of a common year
9083 \ifbbl@hebrleap
9084 \else
9085 \advance #2 by 1
9086 \fi
9087 \fi
9088 \fi
9089 \ifnum \tmpc < 16789
9090 \else
9091 \ifnum \tmpa=1
9092 \advance #1 by -1
9093 \bbl@checkleaphebrewyear{#1}% at the end of leap year
9094 \ifbbl@hebrleap
9095 \advance #2 by 1
9096 \fi
9097 \fi
9098 \fi
9099 \else
9100 \advance #2 by 1
9101 \fi
9102 \bbl@remainder{#2}{7}{\tmpa}%
9103 \ifnum \tmpa=0
9104 \advance #2 by 1
9105 \else
9106 \ifnum \tmpa=3
9107 \advance #2 by 1
9108 \else
9109 \ifnum \tmpa=5
9110 \advance #2 by 1
9111 \fi
9112 \fi
9113 \fi
9114 \global\bbl@cntcommon=#2\relax}%

```

```

9115 #2=\bbl@cntcommon}
9116 \def\bbl@daysinhebrewyear#1#2{%
9117 {\countdef\tmpe=12
9118 \bbl@hebreleapseddays{#1}{\tmpe}%
9119 \advance #1 by 1
9120 \bbl@hebreleapseddays{#1}{#2}%
9121 \advance #2 by -\tmpe
9122 \global\bbl@cntcommon=#2}%
9123 #2=\bbl@cntcommon}
9124 \def\bbl@hebrdayspriormonths#1#2#3{%
9125 {\countdef\tmpf= 14
9126 #3=\ifcase #1
9127 0 \or
9128 0 \or
9129 30 \or
9130 59 \or
9131 89 \or
9132 118 \or
9133 148 \or
9134 148 \or
9135 177 \or
9136 207 \or
9137 236 \or
9138 266 \or
9139 295 \or
9140 325 \or
9141 400
9142 \fi
9143 \bbl@checkleaphebrewyear{#2}%
9144 \ifbbl@hebrleap
9145 \ifnum #1 > 6
9146 \advance #3 by 30
9147 \fi
9148 \fi
9149 \bbl@daysinhebrewyear{#2}{\tmpf}%
9150 \ifnum #1 > 3
9151 \ifnum \tmpf=353
9152 \advance #3 by -1
9153 \fi
9154 \ifnum \tmpf=383
9155 \advance #3 by -1
9156 \fi
9157 \fi
9158 \ifnum #1 > 2
9159 \ifnum \tmpf=355
9160 \advance #3 by 1
9161 \fi
9162 \ifnum \tmpf=385
9163 \advance #3 by 1
9164 \fi
9165 \fi
9166 \global\bbl@cntcommon=#3\relax}%
9167 #3=\bbl@cntcommon}
9168 \def\bbl@absfromhebr#1#2#3#4{%
9169 {#4=#1\relax
9170 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
9171 \advance #4 by #1\relax
9172 \bbl@hebreleapseddays{#3}{#1}%
9173 \advance #4 by #1\relax
9174 \advance #4 by -1373429
9175 \global\bbl@cntcommon=#4\relax}%
9176 #4=\bbl@cntcommon}
9177 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%

```

```

9178 {\countdef\tmpx= 17
9179 \countdef\tmpy= 18
9180 \countdef\tmpz= 19
9181 #6=#3\relax
9182 \global\advance #6 by 3761
9183 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
9184 \tmpz=1 \tmpy=1
9185 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
9186 \ifnum \tmpx > #4\relax
9187     \global\advance #6 by -1
9188     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
9189 \fi
9190 \advance #4 by -\tmpx
9191 \advance #4 by 1
9192 #5=#4\relax
9193 \divide #5 by 30
9194 \loop
9195     \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
9196     \ifnum \tmpx < #4\relax
9197         \advance #5 by 1
9198         \tmpy=\tmpx
9199 \repeat
9200 \global\advance #5 by -1
9201 \global\advance #4 by -\tmpy}}
9202 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
9203 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
9204 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
9205     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
9206     \bbl@hebrfromgreg
9207         {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
9208         {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
9209 \edef#4{\the\bbl@hebryear}%
9210 \edef#5{\the\bbl@hebrmonth}%
9211 \edef#6{\the\bbl@hebrday}}
9212 (/ca-hebrew)

```

12.3. 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).

```

9213 (*ca-persian)
9214 <@Compute Julian day@>
9215 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
9216     2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
9217 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
9218     \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
9219     \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
9220         \bbl@afterfi\expandafter@gobble
9221     \fi\fi
9222     {\bbl@error{year-out-range}{2013-2050}{}}}%
9223 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
9224 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
9225 \edef\bbl@tempc{\fpeval{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
9226 \def\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
9227 \ifnum\bbl@tempc<\bbl@tempb
9228     \edef\bbl@tempa{\fpeval{\bbl@tempa-1}}% go back 1 year and redo
9229     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
9230     \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
9231     \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
9232 \fi

```

```

9233 \edef#4{\fpeval{\bbl@tempa-621}}% set Jalali year
9234 \edef#6{\fpeval{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
9235 \edef#5{\fpeval{% set Jalali month
9236     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
9237 \edef#6{\fpeval{% set Jalali day
9238     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
9239 (/ca-persian)

```

12.4. 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.

```

9240 (*ca-coptic)
9241 <@Compute Julian day@>
9242 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
9243     \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
9244     \edef\bbl@tempc{\fpeval{\bbl@tempd - 1825029.5}}%
9245     \edef#4{\fpeval{%
9246         floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
9247     \edef\bbl@tempc{\fpeval{%
9248         \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
9249     \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
9250     \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
9251 (/ca-coptic)
9252 (*ca-ethiopic)
9253 <@Compute Julian day@>
9254 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
9255     \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
9256     \edef\bbl@tempc{\fpeval{\bbl@tempd - 1724220.5}}%
9257     \edef#4{\fpeval{%
9258         floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
9259     \edef\bbl@tempc{\fpeval{%
9260         \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
9261     \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
9262     \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
9263 (/ca-ethiopic)

```

12.5. Julian

Based on [ReinDersh].

```

9264 (*ca-julian)
9265 <@Compute Julian day@>
9266 \def\bbl@ca@julian#1-#2-#3\@@#4#5#6{%
9267     \edef\bbl@tempj{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + .5}}%
9268     \edef\bbl@tempa{\fpeval{\bbl@tempj + 32082.5}}%
9269     \edef\bbl@tempb{\fpeval{floor((4 * \bbl@tempa + 3) / 1461)}}%
9270     \edef\bbl@tempc{\fpeval{\bbl@tempa - floor(1461*\bbl@tempb/4)}}%
9271     \edef\bbl@tempd{\fpeval{floor((5 * \bbl@tempc + 2) / 153)}}%
9272     \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
9273     \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
9274     \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}}
9275 (/ca-julian)

```

The Indian National Calendar, Saka era.

```

9276 (*ca-indian)
9277 <@Compute Julian day@>
9278 \def\bbl@sakachaitra#1{%
9279     \bbl@cs@jd{#1}{3}%
9280     \fpeval{ ((\bbl@fpmo{#1}{4})==0 && \bbl@fpmo{#1}{100})!=0)
9281     || \bbl@fpmo{#1}{400})==0) ? 21 : 22 }}
9282 \def\bbl@ca@indian#1-#2-#3\@@#4#5#6{%
9283     \edef\bbl@tempa{\fpeval{\bbl@cs@jd{#1}{#2}{#3}}}%

```

```

9284 \edef\bbl@tempb{\fpeval{\bbl@sakachaitra{#1}}}%
9285 \edef\bbl@tempc{\fpeval{\bbl@tempa < \bbl@tempb ? #1-1 : #1}}%
9286 \edef\bbl@tempd{\fpeval{\bbl@tempa-\bbl@sakachaitra{\bbl@tempc}}}%
9287 \edef\bbl@tempe{\fpeval{ ((\bbl@fmod{\bbl@tempc}{4})==0 &&
9288 \bbl@fmod{\bbl@tempc}{100}!=0) || \bbl@fmod{\bbl@tempc}{400}==0)?31:30}}%
9289 \edef#4{\fpeval{\bbl@tempc-78}}%
9290 \edef#5{\fpeval{\bbl@tempd<\bbl@tempe ? 1 :
9291 (\bbl@tempd-\bbl@tempe < 155 ?
9292 2+trunc((\bbl@tempd-\bbl@tempe)/31,0) :
9293 7+trunc((\bbl@tempd-\bbl@tempe-155)/30,0))}}%
9294 \edef#6{\fpeval{\bbl@tempd<\bbl@tempe ? \bbl@tempd+1 :
9295 (\bbl@tempd-\bbl@tempe < 155 ?
9296 \bbl@fmod{\bbl@tempd-\bbl@tempe}{31}+1 :
9297 \bbl@fmod{\bbl@tempd-\bbl@tempe-155}{30}+1)}}}
9298 (/ca-indian)
9299 %

```

12.6. Buddhist

That's very simple.

```

9300 (*ca-buddhist)
9301 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
9302 \edef#4{\number\numexpr#1+543\relax}%
9303 \edef#5{#2}%
9304 \edef#6{#3}}
9305 (/ca-buddhist)
9306 %
9307 % \subsection{Chinese}
9308 %
9309 % Brute force, with the Julian day of first day of each month. The
9310 % table has been computed with the help of \textsf{python-lunardate} by
9311 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
9312 % is 2015-2044.
9313 %
9314 % \begin{macrocode}
9315 (*ca-chinese)
9316 \ExplSyntaxOn
9317 <@Compute Julian day@>
9318 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
9319 \edef\bbl@tempd{\fpeval{%
9320 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
9321 \count@\z@
9322 \@tempcnta=2015
9323 \bbl@foreach\bbl@cs@chinese@data{%
9324 \ifnum##1>\bbl@tempd\else
9325 \advance\count@\@ne
9326 \ifnum\count@>12
9327 \count@\@ne
9328 \advance\@tempcnta\@ne\fi
9329 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
9330 \ifin@
9331 \advance\count@\m@ne
9332 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
9333 \else
9334 \edef\bbl@tempe{\the\count@}%
9335 \fi
9336 \edef\bbl@tempb{##1}%
9337 \fi}%
9338 \edef#4{\the\@tempcnta}%
9339 \edef#5{\bbl@tempe}%
9340 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
9341 \def\bbl@cs@chinese@leap{%
9342 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}

```

```

9343 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
9344 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
9345 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
9346 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
9347 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
9348 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
9349 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
9350 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
9351 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
9352 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
9353 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
9354 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
9355 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
9356 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
9357 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
9358 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
9359 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
9360 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
9361 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
9362 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
9363 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
9364 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
9365 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
9366 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
9367 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
9368 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
9369 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
9370 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
9371 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
9372 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
9373 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
9374 10896,10926,10956,10986,11015,11045,11074,11103}
9375 \ExplSyntaxOff
9376 (/ca-chinese)

```

13. Acknowledgements

In the initial stages of the development of babel, Bernd Raichle provided many helpful suggestions and Michel Goossens supplied contributions for many languages. Ideas from Nico Poppelier, Piet van Oostrum and many others have been used. Paul Wackers and Werenfried Spit helped find and repair bugs.

More recently, there are significant contributions by Salim Bou, Ulrike Fischer, Loren Davis and Udi Fogiel.

Barbara Beeton has helped in improving the manual.

There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national $\TeX$ styles*, *TUGboat* 10 (1989) #3, pp. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *$\TeX$, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.

- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German T_EX*, *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International E_T_EX is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using E_T_EX*, Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij (’s-Gravenhage, 1988).