Babel

Code

Version 24.13
2024/11/08

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and internationalization

Unicode
TeX
pdfTeX
LuaTeX
XeTeX

Contents

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

1. Identification and loading of required files

The babel package after unpacking consists of the following files:

babel.sty is the LaTeX package, which set options and load language styles.
babel.def is loaded by Plain.
switch.def defines macros to set and switch languages (it loads part `babel.def`).
plain.def is not used, and just loads babel.def, for compatibility.
hyphen.cfg is the file to be used when generating the formats to load hyphenation patterns.

There some additional `tex`, `def` and `lua` files.

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

2. `locale` directory

A required component of babel is a set of `ini` files with basic definitions for about 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 (eg, there are no geographic areas in Spanish). Not all include LICR variants.

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

See Keys in ini files in the the babel site.

3. Tools

```
1 ⟨⟨version=24.13⟩⟩
2 ⟨⟨date=2024/11/08⟩⟩
```

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

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

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

\bbl@add@list This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.

```
25 \def\bbl@add@list#1#2{%
26   \edef#1{%
27     \bbl@ifunset{\bbl@stripslash#1}%
28       {}%
29       {\ifx#1\@empty\else#1,\fi}%
30     #2}}
```

\bbl@afterelse

\bbl@afterfi Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the \else and \fi parts of an \if-statement[1]. 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}}
```

---

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

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

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

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

```
76 \def\bbl@ifblank#1{%
77   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
79 \def\bbl@ifset#1#2#3{%
80   \bbl@ifunset{#1}{#3}{\bbl@exp{\\\bbl@ifblank{\@nameuse{#1}}}{#3}{#2}}}
```

For each element in the comma separated <key>=<value> list, execute <code> with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the <key> alone, it passes \@empty (ie, the macro thus named, not an empty argument, which is what you get with <key>= and no value).

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

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

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

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

```
101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
```

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

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

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

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

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

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

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

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

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

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

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

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

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

3.1. A few core definitions

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

```
199 ⟨⟨*Define core switching macros⟩⟩ ≡
200 \ifx\language\@undefined
201  \csname newcount\endcsname\language
202 \fi
203 ⟨⟨/Define core switching macros⟩⟩
```

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

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

```
204 ⟨⟨*Define core switching macros⟩⟩ ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 ⟨⟨/Define core switching macros⟩⟩
```

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

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

3.2. LaTeX: `babel.sty` (start)

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

```
208 ⟨*package⟩
209 \NeedsTeXFormat{LaTeX2e}
210 \ProvidesPackage{babel}%
211   [<@date@> v<@version@> %%NB%%
212    The multilingual framework for pdfLaTeX, LuaLaTeX and XeLaTeX]
```

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

```
213 \@ifpackagewith{babel}{debug}
214   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
215    \let\bbl@debug\@firstofone
216    \ifx\directlua\@undefined\else
217      \directlua{
218        Babel = Babel or {}
219        Babel.debug = true }%
220      \input{babel-debug.tex}%
221    \fi}
222   {\providecommand\bbl@trace[1]{}%
223    \let\bbl@debug\@gobble
224    \ifx\directlua\@undefined\else
225      \directlua{
226        Babel = Babel or {}
227        Babel.debug = false }%
228    \fi}
```

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

```
229 \def\bbl@error#1{% Implicit #2#3#4
230   \begingroup
231     \catcode`\\=0 \catcode`\==12 \catcode`\`=12
232     \input errbabel.def
233   \endgroup
234   \bbl@error{#1}}
235 \def\bbl@warning#1{%
236   \begingroup
237     \def\\{\MessageBreak}%
238     \PackageWarning{babel}{#1}%
239   \endgroup}
240 \def\bbl@infowarn#1{%
241   \begingroup
242     \def\\{\MessageBreak}%
243     \PackageNote{babel}{#1}%
```

```
244   \endgroup}
245 \def\bbl@info#1{%
246   \begingroup
247     \def\\{\MessageBreak}%
248     \PackageInfo{babel}{#1}%
249   \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.

```
250 <@Basic macros@>
251 \@ifpackagewith{babel}{silent}
252   {\let\bbl@info\@gobble
253    \let\bbl@infowarn\@gobble
254    \let\bbl@warning\@gobble}
255   {}
256 %
257 \def\AfterBabelLanguage#1{%
258   \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.

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

```
277 \bbl@trace{Defining option 'base'}
278 \@ifpackagewith{babel}{base}{%
279   \let\bbl@onlyswitch\@empty
280   \let\bbl@provide@locale\relax
281   \input babel.def
282   \let\bbl@onlyswitch\@undefined
283   \ifx\directlua\@undefined
284     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
285   \else
286     \input luababel.def
287     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
288   \fi
289   \DeclareOption{base}{}%
290   \DeclareOption{showlanguages}{}%
291   \ProcessOptions
```

```
292    \global\expandafter\let\csname opt@babel.sty\endcsname\relax
293    \global\expandafter\let\csname ver@babel.sty\endcsname\relax
294    \global\let\@ifl@ter@@\@ifl@ter
295    \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
296    \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.

```
297 \bbl@trace{key=value and another general options}
298 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
299 \def\bbl@tempb#1.#2{%  Remove trailing dot
300    #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
301 \def\bbl@tempe#1=#2\@@{%
302   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
303 \def\bbl@tempd#1.#2\@nnil{%%^^A  TODO. Refactor lists?
304   \ifx\@empty#2%
305     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
306   \else
307     \in@{,provide=}{,#1}%
308     \ifin@
309       \edef\bbl@tempc{%
310         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
311     \else
312       \in@{$modifiers$}{$#1$}%%^^A TODO. Allow spaces.
313       \ifin@
314         \bbl@tempe#2\@@
315       \else
316         \in@{=}{#1}%
317         \ifin@
318           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
319         \else
320           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
321           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
322         \fi
323       \fi
324     \fi
325   \fi}
326 \let\bbl@tempc\@empty
327 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
328 \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.

```
329 \DeclareOption{KeepShorthandsActive}{}
330 \DeclareOption{activeacute}{}
331 \DeclareOption{activegrave}{}
332 \DeclareOption{debug}{}
333 \DeclareOption{noconfigs}{}
334 \DeclareOption{showlanguages}{}
335 \DeclareOption{silent}{}
336 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
337 \chardef\bbl@iniflag\z@
338 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}     % main -> +1
339 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}   % second = 2
340 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
341 % A separate option
342 \let\bbl@autoload@options\@empty
343 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
344 % Don't use. Experimental. TODO.
```

```
345 \newif\ifbbl@single
346 \DeclareOption{selectors=off}{\bbl@singletrue}
347 <@More package options@>
```

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

```
348 \let\bbl@opt@shorthands\@nnil
349 \let\bbl@opt@config\@nnil
350 \let\bbl@opt@main\@nnil
351 \let\bbl@opt@headfoot\@nnil
352 \let\bbl@opt@layout\@nnil
353 \let\bbl@opt@provide\@nnil
```

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

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

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

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

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

```
368 \ProcessOptions*
```

3.5. Post-process some options

```
369 \ifx\bbl@opt@provide\@nnil
370   \let\bbl@opt@provide\@empty  % %%% MOVE above
371 \else
372   \chardef\bbl@iniflag\@ne
373   \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
374     \in@{,provide,}{,#1,}%
375     \ifin@
376       \def\bbl@opt@provide{#2}%
377     \fi}
378 \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=....

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

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

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

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

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

```
400   \edef\bbl@opt@shorthands{%
401     \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.

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

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

```
407 \ifx\bbl@opt@headfoot\@nnil\else
408   \g@addto@macro\@resetactivechars{%
409     \set@typeset@protect
410     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
411     \let\protect\noexpand}
412 \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.

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

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

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

12

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

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

First, exit immediately if previouly loaded.

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

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

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

```
446 ⟨*package | core⟩
447 \def\bbl@version{<@version@>}
448 \def\bbl@date{<@date@>}
449 <@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.

```
450 \def\adddialect#1#2{%
451   \global\chardef#1#2\relax
452   \bbl@usehooks{adddialect}{{#1}{#2}}%
453   \begingroup
454     \count@#1\relax
455     \def\bbl@elt##1##2##3##4{%
456       \ifnum\count@=##2\relax
457         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
458         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
459                   set to \expandafter\string\csname l@##1\endcsname\\%
460                   (\string\language\the\count@). Reported}%
461         \def\bbl@elt####1####2####3####4{}%
462       \fi}%
463     \bbl@cs{languages}%
464   \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.

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

```
477        \@empty
478      \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
479    \bbl@tempd
480    \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}
481 \def\bbl@iflanguage#1{%
482    \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}
```

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

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

```
483 \def\bbl@bcpcase#1#2#3#4\@@#5{%
484   \ifx\@empty#3%
485     \uppercase{\def#5{#1#2}}%
486   \else
487     \uppercase{\def#5{#1}}%
488     \lowercase{\edef#5{#5#2#3#4}}%
489   \fi}
490 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
491   \let\bbl@bcp\relax
492   \lowercase{\def\bbl@tempa{#1}}%
493   \ifx\@empty#2%
494     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
495   \else\ifx\@empty#3%
496     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
497     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
498       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
499       {}%
500     \ifx\bbl@bcp\relax
501       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
502     \fi
503   \else
504     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
505     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
506     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
507       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
508       {}%
509     \ifx\bbl@bcp\relax
510       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
511         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
512         {}%
513     \fi
514     \ifx\bbl@bcp\relax
515       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
516         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
517         {}%
518     \fi
519     \ifx\bbl@bcp\relax
520       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
521     \fi
522   \fi\fi}
523 \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.

```
524 \def\iflanguage#1{%
525   \bbl@iflanguage{#1}{%
526     \ifnum\csname l@#1\endcsname=\language
```

```
527        \expandafter\@firstoftwo
528      \else
529        \expandafter\@secondoftwo
530      \fi}}
```

4.1. Selecting the language

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

```
531 \let\bbl@select@type\z@
532 \edef\selectlanguage{%
533   \noexpand\protect
534   \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.

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

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

```
536 \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.

```
537 \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:

```
538 \def\bbl@push@language{%
539   \ifx\languagename\@undefined\else
540     \ifx\currentgrouplevel\@undefined
541       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
542     \else
543       \ifnum\currentgrouplevel=\z@
544         \xdef\bbl@language@stack{\languagename+}%
545       \else
546         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
547       \fi
548     \fi
549   \fi}
```

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

```

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

```
550 \def\bbl@pop@lang#1+#2\@@{%
551   \edef\languagename{#1}%
552   \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).

```
553 \let\bbl@ifrestoring\@secondoftwo
554 \def\bbl@pop@language{%
555   \expandafter\bbl@pop@lang\bbl@language@stack\@@
556   \let\bbl@ifrestoring\@firstoftwo
557   \expandafter\bbl@set@language\expandafter{\languagename}%
558   \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).

```
559 \chardef\localeid\z@
560 \def\bbl@id@last{0}      % No real need for a new counter
561 \def\bbl@id@assign{%
562   \bbl@ifunset{bbl@id@@\languagename}%
563     {\count@\bbl@id@last\relax
564      \advance\count@\@ne
565      \bbl@csarg\chardef{id@@\languagename}\count@
566      \edef\bbl@id@last{\the\count@}%
567      \ifcase\bbl@engine\or
568        \directlua{
569          Babel.locale_props[\bbl@id@last] = {}
570          Babel.locale_props[\bbl@id@last].name = '\languagename'
571          Babel.locale_props[\bbl@id@last].vars = {}
572        }%
573      \fi}%
574     {}%
575   \chardef\localeid\bbl@cl{id@}}
```

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

```
576 \expandafter\def\csname selectlanguage \endcsname#1{%
577   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
578   \bbl@push@language
579   \aftergroup\bbl@pop@language
580   \bbl@set@language{#1}}
581 \let\endselectlanguage\relax
```

\bbl@set@language   The macro \bbl@set@language takes care of switching the language
environment *and* of writing entries on the auxiliary files. For historical reasons, language names can
be either language of \language. To catch either form a trick is used, but unfortunately as a side
effect the catcodes of letters in \languagename are messed up. This is a bug, but preserved for
backwards compatibility. The list of auxiliary files can be extended by redefining
\BabelContentsFiles, but make sure they are loaded inside a group (as aux, toc, lof, and lot do)
or the last language of the document will remain active afterwards.

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

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

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

Then we have to *re*define \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.

```
628 \newif\ifbbl@usedategroup
629 \let\bbl@savedextras\@empty
```

```
630 \def\bbl@switch#1{%  from select@, foreign@
631 % make sure there is info for the language if so requested
632 \bbl@ensureinfo{#1}%
633 % restore
634 \originalTeX
635 \expandafter\def\expandafter\originalTeX\expandafter{%
636   \csname noextras#1\endcsname
637   \let\originalTeX\@empty
638   \babel@beginsave}%
639 \bbl@usehooks{afterreset}{}%
640 \languageshorthands{none}%
641 % set the locale id
642 \bbl@id@assign
643 % switch captions, date
644 \bbl@bsphack
645   \ifcase\bbl@select@type
646     \csname captions#1\endcsname\relax
647     \csname date#1\endcsname\relax
648   \else
649     \bbl@xin@{,captions,}{,\bbl@select@opts,}%
650     \ifin@
651       \csname captions#1\endcsname\relax
652     \fi
653     \bbl@xin@{,date,}{,\bbl@select@opts,}%
654     \ifin@  % if \foreign... within \<language>date
655       \csname date#1\endcsname\relax
656     \fi
657   \fi
658 \bbl@esphack
659 % switch extras
660 \csname bbl@preextras@#1\endcsname
661 \bbl@usehooks{beforeextras}{}%
662 \csname extras#1\endcsname\relax
663 \bbl@usehooks{afterextras}{}%
664 %  > babel-ensure
665 %  > babel-sh-<short>
666 %  > babel-bidi
667 %  > babel-fontspec
668 \let\bbl@savedextras\@empty
669 % hyphenation - case mapping
670 \ifcase\bbl@opt@hyphenmap\or
671   \def\BabelLower##1##2{\lccode##1=##2\relax}%
672   \ifnum\bbl@hymapsel>4\else
673     \csname\languagename @bbl@hyphenmap\endcsname
674   \fi
675   \chardef\bbl@opt@hyphenmap\z@
676 \else
677   \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
678     \csname\languagename @bbl@hyphenmap\endcsname
679   \fi
680 \fi
681 \let\bbl@hymapsel\@cclv
682 % hyphenation - select rules
683 \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
684   \edef\bbl@tempa{u}%
685 \else
686   \edef\bbl@tempa{\bbl@cl{lnbrk}}%
687 \fi
688 % linebreaking - handle u, e, k (v in the future)
689 \bbl@xin@{/u}{/\bbl@tempa}%
690 \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
691 \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
692 \ifin@\else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
```

18

```
693  \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
694  % hyphenation - save mins
695  \babel@savevariable\lefthyphenmin
696  \babel@savevariable\righthyphenmin
697  \ifnum\bbl@engine=\@ne
698    \babel@savevariable\hyphenationmin
699  \fi
700  \ifin@
701    % unhyphenated/kashida/elongated/padding = allow stretching
702    \language\l@unhyphenated
703    \babel@savevariable\emergencystretch
704    \emergencystretch\maxdimen
705    \babel@savevariable\hbadness
706    \hbadness\@M
707  \else
708    % other = select patterns
709    \bbl@patterns{#1}%
710  \fi
711  % hyphenation - set mins
712  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
713    \set@hyphenmins\tw@\thr@@\relax
714    \@nameuse{bbl@hyphenmins@}%
715  \else
716    \expandafter\expandafter\expandafter\set@hyphenmins
717      \csname #1hyphenmins\endcsname\relax
718  \fi
719  \@nameuse{bbl@hyphenmins@}%
720  \@nameuse{bbl@hyphenmins@\languagename}%
721  \@nameuse{bbl@hyphenatmin@}%
722  \@nameuse{bbl@hyphenatmin@\languagename}%
723  \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.

```
724 \long\def\otherlanguage#1{%
725  \def\bbl@selectorname{other}%
726  \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
727  \csname selectlanguage \endcsname{#1}%
728  \ignorespaces}
```

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

```
729 \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.

```
730 \expandafter\def\csname otherlanguage*\endcsname{%
731  \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
732 \def\bbl@otherlanguage@s[#1]#2{%
733  \def\bbl@selectorname{other*}%
734  \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
735  \def\bbl@select@opts{#1}%
736  \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".

```
737 \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.

```
738 \providecommand\bbl@beforeforeign{}
739 \edef\foreignlanguage{%
740   \noexpand\protect
741   \expandafter\noexpand\csname foreignlanguage \endcsname}
742 \expandafter\def\csname foreignlanguage \endcsname{%
743   \@ifstar\bbl@foreign@s\bbl@foreign@x}
744 \providecommand\bbl@foreign@x[3][]{%
745   \begingroup
746     \def\bbl@selectorname{foreign}%
747     \def\bbl@select@opts{#1}%
748     \let\BabelText\@firstofone
749     \bbl@beforeforeign
750     \foreign@language{#2}%
751     \bbl@usehooks{foreign}{}%
752     \BabelText{#3}% Now in horizontal mode!
753   \endgroup}
754 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
755   \begingroup
756     {\par}%
757     \def\bbl@selectorname{foreign*}%
758     \let\bbl@select@opts\@empty
759     \let\BabelText\@firstofone
760     \foreign@language{#1}%
761     \bbl@usehooks{foreign*}{}%
762     \bbl@dirparastext
763     \BabelText{#2}% Still in vertical mode!
764     {\par}%
765   \endgroup}
766 \providecommand\BabelWrapText[1]{%
767     \def\bbl@tempa{\def\BabelText####1}%
768     \expandafter\bbl@tempa\expandafter{\BabelText{#1}}}
```

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

```
769 \def\foreign@language#1{%
770   % set name
771   \edef\languagename{#1}%
772   \ifbbl@usedategroup
773     \bbl@add\bbl@select@opts{,date,}%
774     \bbl@usedategroupfalse
775   \fi
```

```
776  \bbl@fixname\languagename
777  \let\localename\languagename
778  % TODO. name@map here?
779  \bbl@provide@locale
780  \bbl@iflanguage\languagename{%
781    \let\bbl@select@type\@ne
782    \expandafter\bbl@switch\expandafter{\languagename}}}
```

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

```
783 \def\IfBabelSelectorTF#1{%
784   \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
785   \ifin@
786     \expandafter\@firstoftwo
787   \else
788     \expandafter\@secondoftwo
789   \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.

```
790 \let\bbl@hyphlist\@empty
791 \let\bbl@hyphenation@\relax
792 \let\bbl@pttnlist\@empty
793 \let\bbl@patterns@\relax
794 \let\bbl@hymapsel=\@cclv
795 \def\bbl@patterns#1{%
796   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
797       \csname l@#1\endcsname
798       \edef\bbl@tempa{#1}%
799     \else
800       \csname l@#1:\f@encoding\endcsname
801       \edef\bbl@tempa{#1:\f@encoding}%
802     \fi
803   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
804   %  > luatex
805   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
806     \begingroup
807       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
808       \ifin@\else
809         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
810         \hyphenation{%
811           \bbl@hyphenation@
812           \@ifundefined{bbl@hyphenation@#1}%
813             \@empty
814             {\space\csname bbl@hyphenation@#1\endcsname}}%
815         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
816       \fi
817     \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 otherlanguage*.

```
818 \def\hyphenrules#1{%
819   \edef\bbl@tempf{#1}%
820   \bbl@fixname\bbl@tempf
821   \bbl@iflanguage\bbl@tempf{%
822     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
```

```
823    \ifx\languageshorthands\@undefined\else
824      \languageshorthands{none}%
825    \fi
826    \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
827      \set@hyphenmins\tw@\thr@@\relax
828    \else
829      \expandafter\expandafter\expandafter\set@hyphenmins
830      \csname\bbl@tempf hyphenmins\endcsname\relax
831    \fi}}
832 \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.

```
833 \def\providehyphenmins#1#2{%
834   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
835     \@namedef{#1hyphenmins}{#2}%
836   \fi}
```

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

```
837 \def\set@hyphenmins#1#2{%
838   \lefthyphenmin#1\relax
839   \righthyphenmin#2\relax}
```

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

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

```
840 \ifx\ProvidesFile\@undefined
841   \def\ProvidesLanguage#1[#2 #3 #4]{%
842     \wlog{Language: #1 #4 #3 <#2>}%
843     }
844 \else
845   \def\ProvidesLanguage#1{%
846     \begingroup
847       \catcode`\ 10 %
848       \@makeother\/%
849       \@ifnextchar[%
850         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
851   \def\@provideslanguage#1[#2]{%
852     \wlog{Language: #1 #2}%
853     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
854     \endgroup}
855 \fi
```

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

```
856 \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.

```
857 \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':

```
858 \providecommand\setlocale{\bbl@error{not-yet-available}{}{}{}}
859 \let\uselocale\setlocale
860 \let\locale\setlocale
861 \let\selectlocale\setlocale
862 \let\textlocale\setlocale
863 \let\textlanguage\setlocale
864 \let\languagetext\setlocale
```

4.2. Errors

\@nolanerr

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

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

When the format knows about \PackageError it must be LaTeX 2ε, 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.

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

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

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

4.3. More on selection

\babelensure The user command just parses the optional argument and creates a new macro named \bbl@e@⟨language⟩. We register a hook at the afterextras event which just executes this macro in a "complete" selection (which, if undefined, is \relax and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro \bbl@e@⟨language⟩ contains \bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}, which in 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.

```
896 \bbl@trace{Defining babelensure}
897 \newcommand\babelensure[2][]{%
```

23

```
898  \AddBabelHook{babel-ensure}{afterextras}{%
899    \ifcase\bbl@select@type
900      \bbl@cl{e}%
901    \fi}%
902  \begingroup
903    \let\bbl@ens@include\@empty
904    \let\bbl@ens@exclude\@empty
905    \def\bbl@ens@fontenc{\relax}%
906    \def\bbl@tempb##1{%
907      \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
908    \edef\bbl@tempa{\bbl@tempb#1\@empty}%
909    \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
910    \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
911    \def\bbl@tempc{\bbl@ensure}%
912    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
913      \expandafter{\bbl@ens@include}}%
914    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
915      \expandafter{\bbl@ens@exclude}}%
916    \toks@\expandafter{\bbl@tempc}%
917    \bbl@exp{%
918  \endgroup
919  \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}}
920 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
921   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
922     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
923       \edef##1{\noexpand\bbl@nocaption
924         {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
925     \fi
926     \ifx##1\@empty\else
927       \in@{##1}{#2}%
928       \ifin@\else
929         \bbl@ifunset{bbl@ensure@\languagename}%
930           {\bbl@exp{%
931             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
932               \\\foreignlanguage{\languagename}%
933               {\ifx\relax#3\else
934                 \\\fontencoding{#3}\\\selectfont
935               \fi
936               ########1}}}}%
937           {}%
938         \toks@\expandafter{##1}%
939         \edef##1{%
940           \bbl@csarg\noexpand{ensure@\languagename}%
941           {\the\toks@}}%
942       \fi
943       \expandafter\bbl@tempb
944     \fi}%
945   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
946   \def\bbl@tempa##1{% elt for include list
947     \ifx##1\@empty\else
948       \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
949       \ifin@\else
950         \bbl@tempb##1\@empty
951       \fi
952       \expandafter\bbl@tempa
953     \fi}%
954   \bbl@tempa#1\@empty}
955 \def\bbl@captionslist{%
956   \prefacename\refname\abstractname\bibname\chaptername\appendixname
957   \contentsname\listfigurename\listtablename\indexname\figurename
958   \tablename\partname\enclname\ccname\headtoname\pagename\seename
959   \alsoname\proofname\glossaryname}
```

24

4.4. Short tags

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

```
960 \bbl@trace{Short tags}
961 \newcommand\babeltags[1]{%
962   \edef\bbl@tempa{\zap@space#1 \@empty}%
963   \def\bbl@tempb##1=##2\@@{%
964     \edef\bbl@tempc{%
965       \noexpand\newcommand
966       \expandafter\noexpand\csname ##1\endcsname{%
967         \noexpand\protect
968         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
969       \noexpand\newcommand
970       \expandafter\noexpand\csname text##1\endcsname{%
971         \noexpand\foreignlanguage{##2}}}%
972     \bbl@tempc}%
973   \bbl@for\bbl@tempa\bbl@tempa{%
974     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

4.5. Compatibility with language.def

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

```
975 \bbl@trace{Compatibility with language.def}
976 \ifx\directlua\@undefined\else
977   \ifx\bbl@luapatterns\@undefined
978     \input luababel.def
979   \fi
980 \fi
981 \ifx\bbl@languages\@undefined
982   \ifx\directlua\@undefined
983     \openin1 = language.def % TODO. Remove hardcoded number
984     \ifeof1
985       \closein1
986       \message{I couldn't find the file language.def}
987     \else
988       \closein1
989       \begingroup
990         \def\addlanguage#1#2#3#4#5{%
991           \expandafter\ifx\csname lang@#1\endcsname\relax\else
992             \global\expandafter\let\csname l@#1\expandafter\endcsname
993               \csname lang@#1\endcsname
994           \fi}%
995         \def\uselanguage#1{}%
996         \input language.def
997       \endgroup
998     \fi
999   \fi
1000  \chardef\l@english\z@
1001 \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.

```
1002 \def\addto#1#2{%
1003   \ifx#1\@undefined
1004     \def#1{#2}%
1005   \else
1006     \ifx#1\relax
```

```
1007        \def#1{#2}%
1008      \else
1009        {\toks@\expandafter{#1#2}%
1010         \xdef#1{\the\toks@}}%
1011      \fi
1012    \fi}
```

4.6. Hooks

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

```
1013 \bbl@trace{Hooks}
1014 \newcommand\AddBabelHook[3][]{%
1015   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1016   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1017   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1018   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1019     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1020     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1021   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1022 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1023 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1024 \def\bbl@usehooks{\bbl@usehooks@lang\languagename}
1025 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1026   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1027   \def\bbl@elth##1{%
1028     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@}#3}}%
1029   \bbl@cs{ev@#2@}%
1030   \ifx\languagename\@undefined\else % Test required for Plain (?)
1031     \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1032     \def\bbl@elth##1{%
1033       \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1}#3}}%
1034     \bbl@cs{ev@#2@#1}%
1035   \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).

```
1036 \def\bbl@evargs{,% <- don't delete this comma
1037   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1038   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1039   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1040   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1041   beforestart=0,languagename=2,begindocument=1}
1042 \ifx\NewHook\@undefined\else % Test for Plain (?)
1043   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1044   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1045 \fi
```

Since the following command is meant for a hook (although a LaTeXone), it's placed here.

```
1046 \providecommand\PassOptionsToLocale[2]{%
1047   \bbl@csarg\bbl@add@list{passto@#2}{#1}}
```

4.7. Setting up language files

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

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

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

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

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

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

Finally we check \originalTeX.

```
1048 \bbl@trace{Macros for setting language files up}
1049 \def\bbl@ldfinit{%
1050   \let\bbl@screset\@empty
1051   \let\BabelStrings\bbl@opt@string
1052   \let\BabelOptions\@empty
1053   \let\BabelLanguages\relax
1054   \ifx\originalTeX\@undefined
1055     \let\originalTeX\@empty
1056   \else
1057     \originalTeX
1058   \fi}
1059 \def\LdfInit#1#2{%
1060   \chardef\atcatcode=\catcode`\@
1061   \catcode`\@=11\relax
1062   \chardef\eqcatcode=\catcode`\=
1063   \catcode`\==12\relax
1064   \expandafter\if\expandafter\@backslashchar
1065                 \expandafter\@car\string#2\@nil
1066     \ifx#2\@undefined\else
1067       \ldf@quit{#1}%
1068     \fi
1069   \else
1070     \expandafter\ifx\csname#2\endcsname\relax\else
1071       \ldf@quit{#1}%
1072     \fi
1073   \fi
1074   \bbl@ldfinit}
```

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

```
1075 \def\ldf@quit#1{%
1076   \expandafter\main@language\expandafter{#1}%
1077   \catcode`\@=\atcatcode \let\atcatcode\relax
1078   \catcode`\==\eqcatcode \let\eqcatcode\relax
1079   \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.

```
1080 \def\bbl@afterldf#1{%%^^A TODO. #1 is not used. Remove
1081   \bbl@afterlang
1082   \let\bbl@afterlang\relax
1083   \let\BabelModifiers\relax
1084   \let\bbl@screset\relax}%
1085 \def\ldf@finish#1{%
1086   \loadlocalcfg{#1}%
1087   \bbl@afterldf{#1}%
1088   \expandafter\main@language\expandafter{#1}%
1089   \catcode`\@=\atcatcode \let\atcatcode\relax
1090   \catcode`\==\eqcatcode \let\eqcatcode\relax}
```

After the preamble of the document the commands \LdfInit, \ldf@quit and \ldf@finish are no longer needed. Therefore they are turned into warning messages in LaTeX.

```
1091 \@onlypreamble\LdfInit
1092 \@onlypreamble\ldf@quit
1093 \@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.

```
1094 \def\main@language#1{%
1095   \def\bbl@main@language{#1}%
1096   \let\languagename\bbl@main@language
1097   \let\localename\bbl@main@language
1098   \let\mainlocalename\bbl@main@language
1099   \bbl@id@assign
1100   \bbl@patterns{\languagename}}
```

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

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

```
1101 \def\bbl@beforestart{%
1102   \def\@nolanerr##1{%
1103     \bbl@carg\chardef{l@##1}\z@
1104     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1105   \bbl@usehooks{beforestart}{}%
1106   \global\let\bbl@beforestart\relax}
1107 \AtBeginDocument{%
1108   {\@nameuse{bbl@beforestart}}%  Group!
1109   \if@filesw
1110     \providecommand\babel@aux[2]{}%
1111     \immediate\write\@mainaux{\unexpanded{%
1112       \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1113     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1114   \fi
1115   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1116   \ifbbl@single  % must go after the line above.
1117     \renewcommand\selectlanguage[1]{}%
1118     \renewcommand\foreignlanguage[2]{#2}%
1119     \global\let\babel@aux\@gobbletwo  % Also as flag
1120   \fi}
1121 %
1122 \ifcase\bbl@engine\or
1123   \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1124 \fi
```

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

```
1125 \def\select@language@x#1{%
1126   \ifcase\bbl@select@type
1127     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1128   \else
1129     \select@language{#1}%
1130   \fi}
```

4.8. Shorthands

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

```
1131 \bbl@trace{Shorhands}
1132 \def\bbl@withactive#1#2{%
```

```
1133    \begingroup
1134      \lccode`~=`#2\relax
1135      \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 LaTeX is used). It is used only at one place, namely when \initiate@active@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional.

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

```
1136 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1137   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1138   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1139   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1140     \begingroup
1141       \catcode`#1\active
1142       \nfss@catcodes
1143       \ifnum\catcode`#1=\active
1144         \endgroup
1145         \bbl@add\nfss@catcodes{\@makeother#1}%
1146       \else
1147         \endgroup
1148       \fi
1149   \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 " (ie, with the original "); otherwise \active@char" is executed. This macro in turn expands to \normal@char" in "safe" contexts (eg, \label), but \user@active" in normal "unsafe" ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, \normal@char" is used. However, a deactivated shorthand (with \bbl@deactivate is defined as \active@prefix "\normal@char".

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

```
1150 \def\bbl@active@def#1#2#3#4{%
1151   \@namedef{#3#1}{%
1152     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1153       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1154     \else
1155       \bbl@afterfi\csname#2@sh@#1@\endcsname
1156     \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.

```
1157   \long\@namedef{#3@arg#1}##1{%
1158     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1159       \bbl@afterelse\csname#4#1\endcsname##1%
1160     \else
1161       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1162     \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.

```
1163 \def\initiate@active@char#1{%
1164   \bbl@ifunset{active@char\string#1}%
1165     {\bbl@withactive
1166       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1167     {}}
```

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).

```
1168 \def\@initiate@active@char#1#2#3{%
1169   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1170   \ifx#1\@undefined
1171     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1172   \else
1173     \bbl@csarg\let{oridef@@#2}#1%
1174     \bbl@csarg\edef{oridef@#2}{%
1175       \let\noexpand#1%
1176       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1177   \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*).

```
1178   \ifx#1#3\relax
1179     \expandafter\let\csname normal@char#2\endcsname#3%
1180   \else
1181     \bbl@info{Making #2 an active character}%
1182     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1183       \@namedef{normal@char#2}{%
1184         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1185     \else
1186       \@namedef{normal@char#2}{#3}%
1187     \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).

```
1188     \bbl@restoreactive{#2}%
1189     \AtBeginDocument{%
1190       \catcode`#2\active
1191       \if@filesw
1192         \immediate\write\@mainaux{\catcode`\string#2\active}%
1193       \fi}%
1194     \expandafter\bbl@add@special\csname#2\endcsname
1195     \catcode`#2\active
1196   \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⟩).

```
1197   \let\bbl@tempa\@firstoftwo
1198   \if\string^#2%
1199     \def\bbl@tempa{\noexpand\textormath}%
1200   \else
1201     \ifx\bbl@mathnormal\@undefined\else
1202       \let\bbl@tempa\bbl@mathnormal
1203     \fi
```

```
1204  \fi
1205  \expandafter\edef\csname active@char#2\endcsname{%
1206    \bbl@tempa
1207      {\noexpand\if@safe@actives
1208        \noexpand\expandafter
1209        \expandafter\noexpand\csname normal@char#2\endcsname
1210      \noexpand\else
1211        \noexpand\expandafter
1212        \expandafter\noexpand\csname bbl@doactive#2\endcsname
1213      \noexpand\fi}%
1214    {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1215  \bbl@csarg\edef{doactive#2}{%
1216    \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

$$\texttt{\textbackslash active@prefix} \langle char \rangle \texttt{\textbackslash normal@char} \langle char \rangle$$

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

```
1217  \bbl@csarg\edef{active@#2}{%
1218    \noexpand\active@prefix\noexpand#1%
1219    \expandafter\noexpand\csname active@char#2\endcsname}%
1220  \bbl@csarg\edef{normal@#2}{%
1221    \noexpand\active@prefix\noexpand#1%
1222    \expandafter\noexpand\csname normal@char#2\endcsname}%
1223  \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.

```
1224  \bbl@active@def#2\user@group{user@active}{language@active}%
1225  \bbl@active@def#2\language@group{language@active}{system@active}%
1226  \bbl@active@def#2\system@group{system@active}{normal@char}%
```

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

```
1227  \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1228    {\expandafter\noexpand\csname normal@char#2\endcsname}%
1229  \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1230    {\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@m@s as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```
1231  \if\string'#2%
1232    \let\prim@s\bbl@prim@s
1233    \let\active@math@prime#1%
1234  \fi
1235  \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}}
```

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

```
1236 ⟨⟨*More package options⟩⟩ ≡
1237 \DeclareOption{math=active}{}
1238 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1239 ⟨⟨/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* and the end of the ldf.

```
1240 \@ifpackagewith{babel}{KeepShorthandsActive}%
1241   {\let\bbl@restoreactive\@gobble}%
1242   {\def\bbl@restoreactive#1{%
1243     \bbl@exp{%
1244       \\\AfterBabelLanguage\\\CurrentOption
1245         {\catcode`#1=\the\catcode`#1\relax}%
1246       \\\AtEndOfPackage
1247         {\catcode`#1=\the\catcode`#1\relax}}}%
1248   \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.

```
1249 \def\bbl@sh@select#1#2{%
1250   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1251     \bbl@afterelse\bbl@scndcs
1252   \else
1253     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1254   \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.

```
1255 \begingroup
1256 \bbl@ifunset{ifincsname}%%^^A Ugly. Correct? Only Plain?
1257   {\gdef\active@prefix#1{%
1258     \ifx\protect\@typeset@protect
1259     \else
1260       \ifx\protect\@unexpandable@protect
1261         \noexpand#1%
1262       \else
1263         \protect#1%
1264       \fi
1265       \expandafter\@gobble
1266     \fi}}
1267   {\gdef\active@prefix#1{%
1268     \ifincsname
1269       \string#1%
1270       \expandafter\@gobble
1271     \else
1272       \ifx\protect\@typeset@protect
1273       \else
1274         \ifx\protect\@unexpandable@protect
1275           \noexpand#1%
1276         \else
1277           \protect#1%
1278         \fi
1279         \expandafter\expandafter\expandafter\@gobble
1280       \fi
1281     \fi}}
1282 \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@activestrue), 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).

```
1283 \newif\if@safe@actives
1284 \@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.

```
1285 \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.

```
1286 \chardef\bbl@activated\z@
1287 \def\bbl@activate#1{%
1288   \chardef\bbl@activated\@ne
1289   \bbl@withactive{\expandafter\let\expandafter}#1%
1290     \csname bbl@active@\string#1\endcsname}
1291 \def\bbl@deactivate#1{%
1292   \chardef\bbl@activated\tw@
1293   \bbl@withactive{\expandafter\let\expandafter}#1%
1294     \csname bbl@normal@\string#1\endcsname}
```

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

```
1295 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1296 \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.

```
1297 \def\babel@texpdf#1#2#3#4{%
1298   \ifx\texorpdfstring\@undefined
1299     \textormath{#1}{#3}%
1300   \else
1301     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1302   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1303   \fi}
1304 %
1305 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1306 \def\@decl@short#1#2#3\@nil#4{%
1307   \def\bbl@tempa{#3}%
1308   \ifx\bbl@tempa\@empty
1309     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1310     \bbl@ifunset{#1@sh@\string#2@}{}%
1311       {\def\bbl@tempa{#4}%
1312        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1313        \else
1314          \bbl@info
1315            {Redefining #1 shorthand \string#2\\%
1316             in language \CurrentOption}%
1317        \fi}%
1318     \@namedef{#1@sh@\string#2@}{#4}%
```

```
1319    \else
1320      \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1321      \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1322        {\def\bbl@tempa{#4}%
1323         \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1324         \else
1325           \bbl@info
1326             {Redefining #1 shorthand \string#2\string#3\\%
1327              in language \CurrentOption}%
1328         \fi}%
1329      \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1330    \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.

```
1331 \def\textormath{%
1332   \ifmmode
1333     \expandafter\@secondoftwo
1334   \else
1335     \expandafter\@firstoftwo
1336   \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'.

```
1337 \def\user@group{user}
1338 \def\language@group{english} %^^A I don't like defaults
1339 \def\system@group{system}
```

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

```
1340 \def\useshorthands{%
1341   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1342 \def\bbl@usesh@s#1{%
1343   \bbl@usesh@x
1344     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1345     {#1}}
1346 \def\bbl@usesh@x#1#2{%
1347   \bbl@ifshorthand{#2}%
1348     {\def\user@group{user}%
1349      \initiate@active@char{#2}%
1350      #1%
1351      \bbl@activate{#2}}%
1352     {\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.

```
1353 \def\user@language@group{user@\language@group}
1354 \def\bbl@set@user@generic#1#2{%
1355   \bbl@ifunset{user@generic@active#1}%
1356     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1357      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1358      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1359        \expandafter\noexpand\csname normal@char#1\endcsname}%
```

34

```
1360    \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1361      \expandafter\noexpand\csname user@active#1\endcsname}}%
1362  \@empty}
1363 \newcommand\defineshorthand[3][user]{%
1364  \edef\bbl@tempa{\zap@space#1 \@empty}%
1365  \bbl@for\bbl@tempb\bbl@tempa{%
1366    \if*\expandafter\@car\bbl@tempb\@nil
1367      \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1368      \@expandtwoargs
1369        \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1370    \fi
1371    \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.

```
1372 \def\languageshorthands#1{\def\language@group{#1}}
```

\aliasshorthand *Deprecated*. First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands{"}{/} is \active@prefix /\active@char/, so we still need to let the latter to \active@char".

```
1373 \def\aliasshorthand#1#2{%
1374  \bbl@ifshorthand{#2}%
1375    {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1376        \ifx\document\@notprerr
1377          \@notshorthand{#2}%
1378        \else
1379          \initiate@active@char{#2}%
1380          \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1381          \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1382          \bbl@activate{#2}%
1383        \fi
1384      \fi}%
1385    {\bbl@error{shorthand-is-off}{}{#2}{}}}
```

\@notshorthand

```
1386 \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.

```
1387 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1388 \DeclareRobustCommand*\shorthandoff{%
1389  \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1390 \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.

```
1391 \def\bbl@switch@sh#1#2{%
1392  \ifx#2\@nnil\else
1393    \bbl@ifunset{bbl@active@\string#2}%
1394      {\bbl@error{not-a-shorthand-b}{}{#2}{}}%
1395      {\ifcase#1%  off, on, off*
1396          \catcode`#212\relax
```

```
1397        \or
1398          \catcode`#2\active
1399          \bbl@ifunset{bbl@shdef@\string#2}%
1400            {}%
1401            {\bbl@withactive{\expandafter\let\expandafter}#2%
1402               \csname bbl@shdef@\string#2\endcsname
1403             \bbl@csarg\let{shdef@\string#2}\relax}%
1404          \ifcase\bbl@activated\or
1405            \bbl@activate{#2}%
1406          \else
1407            \bbl@deactivate{#2}%
1408          \fi
1409        \or
1410          \bbl@ifunset{bbl@shdef@\string#2}%
1411            {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1412            {}%
1413          \csname bbl@oricat@\string#2\endcsname
1414          \csname bbl@oridef@\string#2\endcsname
1415        \fi}%
1416      \bbl@afterfi\bbl@switch@sh#1%
1417    \fi}
```

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

```
1418 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1419 \def\bbl@putsh#1{%
1420   \bbl@ifunset{bbl@active@\string#1}%
1421     {\bbl@putsh@i#1\@empty\@nnil}%
1422     {\csname bbl@active@\string#1\endcsname}}
1423 \def\bbl@putsh@i#1#2\@nnil{%
1424   \csname\language@group @sh@\string#1@%
1425     \ifx\@empty#2\else\string#2@\fi\endcsname}
1426 %
1427 \ifx\bbl@opt@shorthands\@nnil\else
1428   \let\bbl@s@initiate@active@char\initiate@active@char
1429   \def\initiate@active@char#1{%
1430     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1431   \let\bbl@s@switch@sh\bbl@switch@sh
1432   \def\bbl@switch@sh#1#2{%
1433     \ifx#2\@nnil\else
1434       \bbl@afterfi
1435       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1436     \fi}
1437   \let\bbl@s@activate\bbl@activate
1438   \def\bbl@activate#1{%
1439     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1440   \let\bbl@s@deactivate\bbl@deactivate
1441   \def\bbl@deactivate#1{%
1442     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1443 \fi
```

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

```
1444 \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.

```
1445 \def\bbl@prim@s{%
1446   \prime\futurelet\@let@token\bbl@pr@m@s}
1447 \def\bbl@if@primes#1#2{%
1448   \ifx#1\@let@token
```

```
1449     \expandafter\@firstoftwo
1450   \else\ifx#2\@let@token
1451     \bbl@afterelse\expandafter\@firstoftwo
1452   \else
1453     \bbl@afterfi\expandafter\@secondoftwo
1454   \fi\fi}
1455 \begingroup
1456   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1457   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1458   \lowercase{%
1459     \gdef\bbl@pr@m@s{%
1460       \bbl@if@primes"'%
1461         \pr@@@s
1462       {\bbl@if@primes*^\pr@@@t\egroup}}}
1463 \endgroup
```

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

```
1464 \initiate@active@char{~}
1465 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1466 \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.

```
1467 \expandafter\def\csname OT1dqpos\endcsname{127}
1468 \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

```
1469 \ifx\f@encoding\@undefined
1470   \def\f@encoding{OT1}
1471 \fi
```

4.9. Language attributes

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

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

```
1472 \bbl@trace{Language attributes}
1473 \newcommand\languageattribute[2]{%
1474   \def\bbl@tempc{#1}%
1475   \bbl@fixname\bbl@tempc
1476   \bbl@iflanguage\bbl@tempc{%
1477     \bbl@vforeach{#2}{%
```

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

```
1478       \ifx\bbl@known@attribs\@undefined
1479         \in@false
1480       \else
1481         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1482       \fi
1483       \ifin@
```

```
1484        \bbl@warning{%
1485          You have more than once selected the attribute '##1'\\%
1486          for language #1. Reported}%
1487      \else
```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated TₑX-code.

```
1488        \bbl@exp{%
1489          \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1490        \edef\bbl@tempa{\bbl@tempc-##1}%
1491        \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1492        {\csname\bbl@tempc @attr@##1\endcsname}%
1493        {\@attrerr{\bbl@tempc}{##1}}%
1494      \fi}}}
1495 \@onlypreamble\languageattribute
```

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

```
1496 \newcommand*{\@attrerr}[2]{%
1497   \bbl@error{unknown-attribute}{#1}{#2}{}}
```

\bbl@declare@ttribute   This command adds the new language/attribute combination to the list of known attributes.
   Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro \extras... for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \begin{document}.

```
1498 \def\bbl@declare@ttribute#1#2#3{%
1499   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1500   \ifin@
1501     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1502   \fi
1503   \bbl@add@list\bbl@attributes{#1-#2}%
1504   \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.

```
1505 \def\bbl@ifattributeset#1#2#3#4{%
1506   \ifx\bbl@known@attribs\@undefined
1507     \in@false
1508   \else
1509     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1510   \fi
1511   \ifin@
1512     \bbl@afterelse#3%
1513   \else
1514     \bbl@afterfi#4%
1515   \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.

```
1516 \def\bbl@ifknown@ttrib#1#2{%
1517   \let\bbl@tempa\@secondoftwo
1518   \bbl@loopx\bbl@tempb{#2}{%
1519     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1520     \ifin@
1521       \let\bbl@tempa\@firstoftwo
```

```
1522    \else
1523    \fi}%
1524  \bbl@tempa}
```

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

```
1525 \def\bbl@clear@ttribs{%
1526   \ifx\bbl@attributes\@undefined\else
1527     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1528       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1529     \let\bbl@attributes\@undefined
1530   \fi}
1531 \def\bbl@clear@ttrib#1-#2.{%
1532   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1533 \AtBeginDocument{\bbl@clear@ttribs}
```

4.10. Support for saving and redefining macros

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

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

```
1534 \bbl@trace{Macros for saving definitions}
1535 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1536 \newcount\babel@savecnt
1537 \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.

```
1538 \def\babel@save#1{%
1539   \def\bbl@tempa{{,#1,}}% Clumsy, for Plain
1540   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1541     \expandafter{\expandafter,\bbl@savedextras,}}%
1542   \expandafter\in@\bbl@tempa
1543   \ifin@\else
1544     \bbl@add\bbl@savedextras{,#1,}%
1545     \bbl@carg\let{babel@\number\babel@savecnt}#1\relax
1546     \toks@\expandafter{\originalTeX\let#1=}%
1547     \bbl@exp{%
1548       \def\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1549     \advance\babel@savecnt\@ne
1550   \fi}
1551 \def\babel@savevariable#1{%
1552   \toks@\expandafter{\originalTeX #1=}%
1553   \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 LaTeX macros completely in case their definitions change (they have changed in the past). A macro named \macro will be saved new control sequences named \org@macro.

```
1554 \def\bbl@redefine#1{%
1555   \edef\bbl@tempa{\bbl@stripslash#1}%
1556   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1557   \expandafter\def\csname\bbl@tempa\endcsname}
1558 \@onlypreamble\bbl@redefine
```

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

```
1559 \def\bbl@redefine@long#1{%
1560   \edef\bbl@tempa{\bbl@stripslash#1}%
1561   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1562   \long\expandafter\def\csname\bbl@tempa\endcsname}
1563 \@onlypreamble\bbl@redefine@long
```

\bbl@redefinerobust   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␣.

```
1564 \def\bbl@redefinerobust#1{%
1565   \edef\bbl@tempa{\bbl@stripslash#1}%
1566   \bbl@ifunset{\bbl@tempa\space}%
1567     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1568      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1569     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1570   \@namedef{\bbl@tempa\space}}
1571 \@onlypreamble\bbl@redefinerobust
```

4.11. French spacing

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

```
1572 \def\bbl@frenchspacing{%
1573   \ifnum\the\sfcode`\.=\@m
1574     \let\bbl@nonfrenchspacing\relax
1575   \else
1576     \frenchspacing
1577     \let\bbl@nonfrenchspacing\nonfrenchspacing
1578   \fi}
1579 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

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

```
1580 \let\bbl@elt\relax
1581 \edef\bbl@fs@chars{%
1582   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1583   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1584   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1585 \def\bbl@pre@fs{%
1586   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1587   \edef\bbl@save@sfcodes{\bbl@fs@chars}}
1588 \def\bbl@post@fs{%
1589   \bbl@save@sfcodes
1590   \edef\bbl@tempa{\bbl@cl{frspc}}%
1591   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
```

```
1592  \if u\bbl@tempa          % do nothing
1593  \else\if n\bbl@tempa      % non french
1594    \def\bbl@elt##1##2##3{%
1595      \ifnum\sfcode`##1=##2\relax
1596        \babel@savevariable{\sfcode`##1}%
1597        \sfcode`##1=##3\relax
1598      \fi}%
1599    \bbl@fs@chars
1600  \else\if y\bbl@tempa      % french
1601    \def\bbl@elt##1##2##3{%
1602      \ifnum\sfcode`##1=##3\relax
1603        \babel@savevariable{\sfcode`##1}%
1604        \sfcode`##1=##2\relax
1605      \fi}%
1606    \bbl@fs@chars
1607  \fi\fi\fi}
```

4.12. Hyphens

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

```
1608  \bbl@trace{Hyphens}
1609  \@onlypreamble\babelhyphenation
1610  \AtEndOfPackage{%
1611    \newcommand\babelhyphenation[2][\@empty]{%
1612      \ifx\bbl@hyphenation@\relax
1613        \let\bbl@hyphenation@\@empty
1614      \fi
1615      \ifx\bbl@hyphlist\@empty\else
1616        \bbl@warning{%
1617          You must not intermingle \string\selectlanguage\space and\\%
1618          \string\babelhyphenation\space or some exceptions will not\\%
1619          be taken into account. Reported}%
1620      \fi
1621      \ifx\@empty#1%
1622        \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1623      \else
1624        \bbl@vforeach{#1}{%
1625          \def\bbl@tempa{##1}%
1626          \bbl@fixname\bbl@tempa
1627          \bbl@iflanguage\bbl@tempa{%
1628            \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1629              \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1630                {}%
1631                {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1632              #2}}}%
1633      \fi}}
```

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

```
1634  \ifx\NewDocumentCommand\@undefined\else
1635    \NewDocumentCommand\babelhyphenmins{sommo}{%
1636      \IfNoValueTF{#2}%
1637        {\protected@edef\bbl@hyphenmins@{\set@hyphenmins{#3}{#4}}%
1638        \IfValueT{#5}{%
1639          \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1640        \IfBooleanT{#1}{%
1641          \lefthyphenmin=#3\relax
1642          \righthyphenmin=#4\relax
1643          \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1644        {\edef\bbl@tempb{\zap@space#2 \@empty}%
```

```
1645      \bbl@for\bbl@tempa\bbl@tempb{%
1646        \@namedef{bbl@hyphenmins@\bbl@tempa}{\set@hyphenmins{#3}{#4}}%
1647        \IfValueT{#5}{%
1648          \@namedef{bbl@hyphenatmin@\bbl@tempa}{\hyphenationmin=#5\relax}}}%
1649      \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}{}{}}}}
1650 \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.

```
1651 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1652 \def\bbl@t@one{T1}
1653 \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.

```
1654 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1655 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1656 \def\bbl@hyphen{%
1657   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1658 \def\bbl@hyphen@i#1#2{%
1659   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1660     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1661     {\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.

```
1662 \def\bbl@usehyphen#1{%
1663   \leavevmode
1664   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1665   \nobreak\hskip\z@skip}
1666 \def\bbl@@usehyphen#1{%
1667   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1668 \def\bbl@hyphenchar{%
1669   \ifnum\hyphenchar\font=\m@ne
1670     \babelnullhyphen
1671   \else
1672     \char\hyphenchar\font
1673   \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.

```
1674 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1675 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1676 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1677 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1678 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1679 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1680 \def\bbl@hy@repeat{%
1681   \bbl@usehyphen{%
1682     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1683 \def\bbl@hy@@repeat{%
1684   \bbl@@usehyphen{%
1685     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
```

```
1686 \def\bbl@hy@empty{\hskip\z@skip}
1687 \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.

```
1688 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

4.13. Multiencoding strings

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

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

```
1689 \bbl@trace{Multiencoding strings}
1690 \def\bbl@toglobal#1{\global\let#1#1}
```

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

```
1691 ⟨⟨*More package options⟩⟩ ≡
1692 \DeclareOption{nocase}{}
1693 ⟨⟨/More package options⟩⟩
```

The following package options control the behavior of \SetString.

```
1694 ⟨⟨*More package options⟩⟩ ≡
1695 \let\bbl@opt@strings\@nnil % accept strings=value
1696 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1697 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1698 \def\BabelStringsDefault{generic}
1699 ⟨⟨/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.

```
1700 \@onlypreamble\StartBabelCommands
1701 \def\StartBabelCommands{%
1702   \begingroup
1703   \@tempcnta="7F
1704   \def\bbl@tempa{%
1705     \ifnum\@tempcnta>"FF\else
1706       \catcode\@tempcnta=11
1707       \advance\@tempcnta\@ne
1708       \expandafter\bbl@tempa
1709     \fi}%
1710   \bbl@tempa
1711   <@Macros local to BabelCommands@>
1712   \def\bbl@provstring##1##2{%
1713     \providecommand##1{##2}%
1714     \bbl@toglobal##1}%
1715   \global\let\bbl@scafter\@empty
1716   \let\StartBabelCommands\bbl@startcmds
1717   \ifx\BabelLanguages\relax
1718     \let\BabelLanguages\CurrentOption
1719   \fi
1720   \begingroup
1721   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1722   \StartBabelCommands}
1723 \def\bbl@startcmds{%
1724   \ifx\bbl@screset\@nnil\else
1725     \bbl@usehooks{stopcommands}{}%
1726   \fi
1727   \endgroup
```

```
1728   \begingroup
1729   \@ifstar
1730     {\ifx\bbl@opt@strings\@nnil
1731        \let\bbl@opt@strings\BabelStringsDefault
1732      \fi
1733      \bbl@startcmds@i}%
1734      \bbl@startcmds@i}
1735 \def\bbl@startcmds@i#1#2{%
1736   \edef\bbl@L{\zap@space#1 \@empty}%
1737   \edef\bbl@G{\zap@space#2 \@empty}%
1738   \bbl@startcmds@ii}
1739 \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 (ie, fallback values). With labelled blocks and `strings=encoded`, define the strings, but with another value, define strings only if the current label or font encoding is the value of `strings`; otherwise (ie, no `strings` or a block whose label is not in `strings=`) do nothing.

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

```
1740 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1741   \let\SetString\@gobbletwo
1742   \let\bbl@stringdef\@gobbletwo
1743   \let\AfterBabelCommands\@gobble
1744   \ifx\@empty#1%
1745     \def\bbl@sc@label{generic}%
1746     \def\bbl@encstring##1##2{%
1747       \ProvideTextCommandDefault##1{##2}%
1748       \bbl@toglobal##1%
1749       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1750     \let\bbl@sctest\in@true
1751   \else
1752     \let\bbl@sc@charset\space % <- zapped below
1753     \let\bbl@sc@fontenc\space % <-    "       "
1754     \def\bbl@tempa##1=##2\@nil{%
1755       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1756     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1757     \def\bbl@tempa##1 ##2{% space -> comma
1758       ##1%
1759       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1760     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1761     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1762     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1763     \def\bbl@encstring##1##2{%
1764       \bbl@foreach\bbl@sc@fontenc{%
1765         \bbl@ifunset{T@####1}%
1766           {}%
1767           {\ProvideTextCommand##1{####1}{##2}%
1768            \bbl@toglobal##1%
1769            \expandafter
1770            \bbl@toglobal\csname####1\string##1\endcsname}}}%
1771     \def\bbl@sctest{%
1772       \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1773   \fi
1774   \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1775   \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
1776     \let\AfterBabelCommands\bbl@aftercmds
1777     \let\SetString\bbl@setstring
1778     \let\bbl@stringdef\bbl@encstring
1779   \else        % ie, strings=value
1780     \bbl@sctest
```

```
1781    \ifin@
1782      \let\AfterBabelCommands\bbl@aftercmds
1783      \let\SetString\bbl@setstring
1784      \let\bbl@stringdef\bbl@provstring
1785    \fi\fi\fi
1786    \bbl@scswitch
1787    \ifx\bbl@G\@empty
1788      \def\SetString##1##2{%
1789        \bbl@error{missing-group}{##1}{}{}}%
1790    \fi
1791    \ifx\@empty#1%
1792      \bbl@usehooks{defaultcommands}{}%
1793    \else
1794      \@expandtwoargs
1795      \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1796    \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) .

```
1797 \def\bbl@forlang#1#2{%
1798    \bbl@for#1\bbl@L{%
1799      \bbl@xin@{,#1,}{,\BabelLanguages,}%
1800      \ifin@#2\relax\fi}}
1801 \def\bbl@scswitch{%
1802    \bbl@forlang\bbl@tempa{%
1803      \ifx\bbl@G\@empty\else
1804        \ifx\SetString\@gobbletwo\else
1805          \edef\bbl@GL{\bbl@G\bbl@tempa}%
1806          \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1807          \ifin@\else
1808            \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1809            \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1810          \fi
1811        \fi
1812      \fi}}
1813 \AtEndOfPackage{%
1814    \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1815    \let\bbl@scswitch\relax}
1816 \@onlypreamble\EndBabelCommands
1817 \def\EndBabelCommands{%
1818    \bbl@usehooks{stopcommands}{}%
1819    \endgroup
1820    \endgroup
1821    \bbl@scafter}
1822 \let\bbl@endcommands\EndBabelCommands
```

Now we define commands to be used inside \StartBabelCommands.

Strings The following macro is the actual definition of \SetString when it is "active"

First save the "switcher". Create it if undefined. Strings are defined only if undefined (ie, like \providescommmand). 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.

```
1823 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1824    \bbl@forlang\bbl@tempa{%
1825      \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1826      \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
```

45

```
1827      {\bbl@exp{%
1828         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
1829      {}%
1830    \def\BabelString{#2}%
1831    \bbl@usehooks{stringprocess}{}%
1832    \expandafter\bbl@stringdef
1833      \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}
```

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

```
1834 \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.

```
1835 ⟨⟨*Macros  local  to  BabelCommands⟩⟩ ≡
1836 \def\SetStringLoop##1##2{%
1837    \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1838    \count@\z@
1839    \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1840      \advance\count@\@ne
1841      \toks@\expandafter{\bbl@tempa}%
1842      \bbl@exp{%
1843        \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1844        \count@=\the\count@\relax}}}%
1845 ⟨⟨/Macros  local  to  BabelCommands⟩⟩
```

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

```
1846 \def\bbl@aftercmds#1{%
1847   \toks@\expandafter{\bbl@scafter#1}%
1848   \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.

```
1849 ⟨⟨*Macros  local  to  BabelCommands⟩⟩ ≡
1850   \newcommand\SetCase[3][]{%
1851    \def\bbl@tempa####1####2{%
1852      \ifx####1\@empty\else
1853        \bbl@carg\bbl@add{extras\CurrentOption}{%
1854          \bbl@carg\babel@save{c__text_uppercase_\string####1_tl}%
1855          \bbl@carg\def{c__text_uppercase_\string####1_tl}{####2}%
1856          \bbl@carg\babel@save{c__text_lowercase_\string####2_tl}%
1857          \bbl@carg\def{c__text_lowercase_\string####2_tl}{####1}}%
1858        \expandafter\bbl@tempa
1859      \fi}%
1860    \bbl@tempa##1\@empty\@empty
1861    \bbl@carg\bbl@toglobal{extras\CurrentOption}}%
1862 ⟨⟨/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.

```
1863 ⟨⟨*Macros  local  to  BabelCommands⟩⟩ ≡
1864   \newcommand\SetHyphenMap[1]{%
1865    \bbl@forlang\bbl@tempa{%
1866      \expandafter\bbl@stringdef
1867        \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1868 ⟨⟨/Macros  local  to  BabelCommands⟩⟩
```

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

```
1869 \newcommand\BabelLower[2]{% one to one.
1870   \ifnum\lccode#1=#2\else
```

```
1871    \babel@savevariable{\lccode#1}%
1872    \lccode#1=#2\relax
1873  \fi}
1874 \newcommand\BabelLowerMM[4]{% many-to-many
1875  \@tempcnta=#1\relax
1876  \@tempcntb=#4\relax
1877  \def\bbl@tempa{%
1878    \ifnum\@tempcnta>#2\else
1879      \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1880      \advance\@tempcnta#3\relax
1881      \advance\@tempcntb#3\relax
1882      \expandafter\bbl@tempa
1883    \fi}%
1884  \bbl@tempa}
1885 \newcommand\BabelLowerMO[4]{% many-to-one
1886  \@tempcnta=#1\relax
1887  \def\bbl@tempa{%
1888    \ifnum\@tempcnta>#2\else
1889      \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1890      \advance\@tempcnta#3
1891      \expandafter\bbl@tempa
1892    \fi}%
1893  \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1894 ⟨⟨*More package options⟩⟩ ≡
1895 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1896 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1897 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1898 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1899 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1900 ⟨⟨/More package options⟩⟩
```

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

```
1901 \AtEndOfPackage{%
1902  \ifx\bbl@opt@hyphenmap\@undefined
1903    \bbl@xin@{,}{\bbl@language@opts}%
1904    \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1905  \fi}
```

4.14. Tailor captions

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

```
1906 \newcommand\setlocalecaption{%%^^A Catch typos.
1907  \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1908 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1909  \bbl@trim@def\bbl@tempa{#2}%
1910  \bbl@xin@{.template}{\bbl@tempa}%
1911  \ifin@
1912    \bbl@ini@captions@template{#3}{#1}%
1913  \else
1914    \edef\bbl@tempd{%
1915      \expandafter\expandafter\expandafter
1916      \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1917    \bbl@xin@
1918      {\expandafter\string\csname #2name\endcsname}%
1919      {\bbl@tempd}%
1920    \ifin@ % Renew caption
1921      \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
1922      \ifin@
1923        \bbl@exp{%
1924          \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
```

```
1925          {\\\bbl@scset\<#2name>\<#1#2name>}%
1926          {}}%
1927       \else % Old way converts to new way
1928        \bbl@ifunset{#1#2name}%
1929          {\bbl@exp{%
1930            \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1931            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
1932              {\def\<#2name>{\<#1#2name>}}%
1933              {}}}%
1934          {}%
1935       \fi
1936     \else
1937       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1938       \ifin@ % New way
1939         \bbl@exp{%
1940           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
1941           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
1942             {\\\bbl@scset\<#2name>\<#1#2name>}%
1943             {}}%
1944       \else  % Old way, but defined in the new way
1945         \bbl@exp{%
1946           \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1947           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
1948             {\def\<#2name>{\<#1#2name>}}%
1949             {}}%
1950       \fi%
1951     \fi
1952     \@namedef{#1#2name}{#3}%
1953     \toks@\expandafter{\bbl@captionslist}%
1954     \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
1955     \ifin@\else
1956       \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
1957       \bbl@toglobal\bbl@captionslist
1958     \fi
1959   \fi}
1960 %^^A \def\bbl@setcaption@s#1#2#3{} % Not yet implemented (w/o 'name')
```

4.15. Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be 'faked', or that are not accessible through T1enc.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.

```
1961 \bbl@trace{Macros related to glyphs}
1962 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1963    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1964    \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.

```
1965 \def\save@sf@q#1{\leavevmode
1966   \begingroup
1967     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1968   \endgroup}
```

4.15.1. Quotation marks

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

```
1969 \ProvideTextCommand{\quotedblbase}{OT1}{%
```

```
1970    \save@sf@q{\set@low@box{\textquotedblright\/}%
1971      \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.

```
1972 \ProvideTextCommandDefault{\quotedblbase}{%
1973   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1974 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1975   \save@sf@q{\set@low@box{\textquoteright\/}%
1976     \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.

```
1977 \ProvideTextCommandDefault{\quotesinglbase}{%
1978   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
1979 \ProvideTextCommand{\guillemetleft}{OT1}{%
1980   \ifmmode
1981     \ll
1982   \else
1983     \save@sf@q{\nobreak
1984       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1985   \fi}
1986 \ProvideTextCommand{\guillemetright}{OT1}{%
1987   \ifmmode
1988     \gg
1989   \else
1990     \save@sf@q{\nobreak
1991       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1992   \fi}
1993 \ProvideTextCommand{\guillemotleft}{OT1}{%
1994   \ifmmode
1995     \ll
1996   \else
1997     \save@sf@q{\nobreak
1998       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1999   \fi}
2000 \ProvideTextCommand{\guillemotright}{OT1}{%
2001   \ifmmode
2002     \gg
2003   \else
2004     \save@sf@q{\nobreak
2005       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2006   \fi}
```

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

```
2007 \ProvideTextCommandDefault{\guillemetleft}{%
2008   \UseTextSymbol{OT1}{\guillemetleft}}
2009 \ProvideTextCommandDefault{\guillemetright}{%
2010   \UseTextSymbol{OT1}{\guillemetright}}
2011 \ProvideTextCommandDefault{\guillemotleft}{%
2012   \UseTextSymbol{OT1}{\guillemotleft}}
2013 \ProvideTextCommandDefault{\guillemotright}{%
2014   \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft

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

```
2015 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2016   \ifmmode
2017     <%
2018   \else
2019     \save@sf@q{\nobreak
2020       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2021   \fi}
2022 \ProvideTextCommand{\guilsinglright}{OT1}{%
2023   \ifmmode
2024     >%
2025   \else
2026     \save@sf@q{\nobreak
2027       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2028   \fi}
```

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

```
2029 \ProvideTextCommandDefault{\guilsinglleft}{%
2030   \UseTextSymbol{OT1}{\guilsinglleft}}
2031 \ProvideTextCommandDefault{\guilsinglright}{%
2032   \UseTextSymbol{OT1}{\guilsinglright}}
```

4.15.2. Letters

\ij

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

```
2033 \DeclareTextCommand{\ij}{OT1}{%
2034   i\kern-0.02em\bbl@allowhyphens j}
2035 \DeclareTextCommand{\IJ}{OT1}{%
2036   I\kern-0.02em\bbl@allowhyphens J}
2037 \DeclareTextCommand{\ij}{T1}{\char188}
2038 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2039 \ProvideTextCommandDefault{\ij}{%
2040   \UseTextSymbol{OT1}{\ij}}
2041 \ProvideTextCommandDefault{\IJ}{%
2042   \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).

```
2043 \def\crrtic@{\hrule height0.1ex width0.3em}
2044 \def\crttic@{\hrule height0.1ex width0.33em}
2045 \def\ddj@{%
2046   \setbox0\hbox{d}\dimen@=\ht0
2047   \advance\dimen@1ex
2048   \dimen@.45\dimen@
2049   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2050   \advance\dimen@ii.5ex
2051   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2052 \def\DDJ@{%
2053   \setbox0\hbox{D}\dimen@=.55\ht0
2054   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2055   \advance\dimen@ii.15ex %            correction for the dash position
2056   \advance\dimen@ii-.15\fontdimen7\font %    correction for cmtt font
2057   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2058   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2059 %
```

```
2060 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2061 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2062 \ProvideTextCommandDefault{\dj}{%
2063   \UseTextSymbol{OT1}{\dj}}
2064 \ProvideTextCommandDefault{\DJ}{%
2065   \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.

```
2066 \DeclareTextCommand{\SS}{OT1}{SS}
2067 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

4.15.3. Shorthands for quotation marks

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

\glq
\grq The 'german' single quotes.

```
2068 \ProvideTextCommandDefault{\glq}{%
2069   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2070 \ProvideTextCommand{\grq}{T1}{%
2071   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2072 \ProvideTextCommand{\grq}{TU}{%
2073   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2074 \ProvideTextCommand{\grq}{OT1}{%
2075   \save@sf@q{\kern-.0125em
2076     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2077     \kern.07em\relax}}
2078 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq
\grqq The 'german' double quotes.

```
2079 \ProvideTextCommandDefault{\glqq}{%
2080   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2081 \ProvideTextCommand{\grqq}{T1}{%
2082   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2083 \ProvideTextCommand{\grqq}{TU}{%
2084   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2085 \ProvideTextCommand{\grqq}{OT1}{%
2086   \save@sf@q{\kern-.07em
2087     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2088     \kern.07em\relax}}
2089 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq
\frq The 'french' single guillemets.

```
2090 \ProvideTextCommandDefault{\flq}{%
2091   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2092 \ProvideTextCommandDefault{\frq}{%
2093   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq

\frqq The 'french' double guillemets.

```
2094 \ProvideTextCommandDefault{\flqq}{%
2095   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2096 \ProvideTextCommandDefault{\frqq}{%
2097   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

4.15.4. Umlauts and tremas

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

\umlauthigh

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

```
2098 \def\umlauthigh{%
2099   \def\bbl@umlauta##1{\leavevmode\bgroup%
2100     \accent\csname\f@encoding dqpos\endcsname
2101       ##1\bbl@allowhyphens\egroup}%
2102   \let\bbl@umlaute\bbl@umlauta}
2103 \def\umlautlow{%
2104   \def\bbl@umlauta{\protect\lower@umlaut}}
2105 \def\umlautelow{%
2106   \def\bbl@umlaute{\protect\lower@umlaut}}
2107 \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.

```
2108 \expandafter\ifx\csname U@D\endcsname\relax
2109   \csname newdimen\endcsname\U@D
2110 \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.

```
2111 \def\lower@umlaut#1{%
2112   \leavevmode\bgroup
2113     \U@D 1ex%
2114     {\setbox\z@\hbox{%
2115       \char\csname\f@encoding dqpos\endcsname}%
2116       \dimen@ -.45ex\advance\dimen@\ht\z@
2117       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2118     \accent\csname\f@encoding dqpos\endcsname
2119     \fontdimen5\font\U@D #1%
2120   \egroup}
```

For all vowels we declare \" to be a composite command which uses \bbl@umlauta or \bbl@umlaute to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package fontenc with option OT1 is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but babel sets them for *all* languages – you may want to redefine \bbl@umlauta and/or \bbl@umlaute for a language in the corresponding ldf (using the babel switching mechanism, of course).

```
2121 \AtBeginDocument{%
2122   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2123   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2124   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
```

```
2125  \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2126  \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2127  \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2128  \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2129  \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2130  \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2131  \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2132  \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}}
```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty \language is defined. Currently used in Amharic.

```
2133 \ifx\l@english\@undefined
2134   \chardef\l@english\z@
2135 \fi
2136 % The following is used to cancel rules in ini files (see Amharic).
2137 \ifx\l@unhyphenated\@undefined
2138   \newlanguage\l@unhyphenated
2139 \fi
```

4.16. Layout

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

```
2140 \bbl@trace{Bidi layout}
2141 \providecommand\IfBabelLayout[3]{#3}%
```

4.17. Load engine specific macros

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

```
2142 \bbl@trace{Input engine specific macros}
2143 \ifcase\bbl@engine
2144   \input txtbabel.def
2145 \or
2146   \input luababel.def
2147 \or
2148   \input xebabel.def
2149 \fi
2150 \providecommand\babelfont{\bbl@error{only-lua-xe}{}{}{}}
2151 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}{}{}}
2152 \ifx\babelposthyphenation\@undefined
2153   \let\babelposthyphenation\babelprehyphenation
2154   \let\babelpatterns\babelprehyphenation
2155   \let\babelcharproperty\babelprehyphenation
2156 \fi
2157 ⟨/package | core⟩
```

4.18. Creating and modifying languages

Continue with LaTeX only.

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

```
2158 ⟨*package⟩
2159 \bbl@trace{Creating languages and reading ini files}
2160 \let\bbl@extend@ini\@gobble
2161 \newcommand\babelprovide[2][]{%
2162   \let\bbl@savelangname\languagename
2163   \edef\bbl@savelocaleid{\the\localeid}%
2164   % Set name and locale id
2165   \edef\languagename{#2}%
2166   \bbl@id@assign
2167   % Initialize keys
```

```
2168  \bbl@vforeach{captions,date,import,main,script,language,%
2169      hyphenrules,linebreaking,justification,mapfont,maparabic,%
2170      mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%
2171      Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2172    {\bbl@csarg\let{KVP@##1}\@nnil}%
2173  \global\let\bbl@release@transforms\@empty
2174  \global\let\bbl@release@casing\@empty
2175  \let\bbl@calendars\@empty
2176  \global\let\bbl@inidata\@empty
2177  \global\let\bbl@extend@ini\@gobble
2178  \global\let\bbl@included@inis\@empty
2179  \gdef\bbl@key@list{;}%
2180  \bbl@ifunset{bbl@passto@#2}%
2181    {\def\bbl@tempa{#1}}%
2182    {\bbl@exp{\def\\\bbl@tempa{\[bbl@passto@#2],\unexpanded{#1}}}}%
2183  \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2184    \in@{/}{##1}% With /, (re)sets a value in the ini
2185    \ifin@
2186      \global\let\bbl@extend@ini\bbl@extend@ini@aux
2187      \bbl@renewinikey##1\@@{##2}%
2188    \else
2189      \bbl@csarg\ifx{KVP@##1}\@nnil\else
2190        \bbl@error{unknown-provide-key}{##1}{}{}%
2191      \fi
2192      \bbl@csarg\def{KVP@##1}{##2}%
2193    \fi}%
2194  \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2195    \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2196  % == init ==
2197  \ifx\bbl@screset\@undefined
2198    \bbl@ldfinit
2199  \fi
2200  % ==
2201  \ifx\bbl@KVP@@import\@nnil\else \ifx\bbl@KVP@import\@nnil
2202    \def\bbl@KVP@import{\@empty}%
2203  \fi\fi
2204  % == date (as option) ==
2205  % \ifx\bbl@KVP@date\@nnil\else
2206  % \fi
2207  % ==
2208  \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2209  \ifcase\bbl@howloaded
2210    \let\bbl@lbkflag\@empty % new
2211  \else
2212    \ifx\bbl@KVP@hyphenrules\@nnil\else
2213      \let\bbl@lbkflag\@empty
2214    \fi
2215    \ifx\bbl@KVP@import\@nnil\else
2216      \let\bbl@lbkflag\@empty
2217    \fi
2218  \fi
2219  % == import, captions ==
2220  \ifx\bbl@KVP@import\@nnil\else
2221    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2222      {\ifx\bbl@initoload\relax
2223        \begingroup
2224          \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2225          \bbl@input@texini{#2}%
2226        \endgroup
2227      \else
2228        \xdef\bbl@KVP@import{\bbl@initoload}%
2229      \fi}%
2230      {}%
```

```
2231    \let\bbl@KVP@date\@empty
2232  \fi
2233  \let\bbl@KVP@captions@@\bbl@KVP@captions %^^A A dirty hack
2234  \ifx\bbl@KVP@captions\@nnil
2235    \let\bbl@KVP@captions\bbl@KVP@import
2236  \fi
2237  % ==
2238  \ifx\bbl@KVP@transforms\@nnil\else
2239    \bbl@replace\bbl@KVP@transforms{ }{,}%
2240  \fi
2241  % == Load ini ==
2242  \ifcase\bbl@howloaded
2243    \bbl@provide@new{#2}%
2244  \else
2245    \bbl@ifblank{#1}%
2246      {}%  With \bbl@load@basic below
2247      {\bbl@provide@renew{#2}}%
2248  \fi
2249  % == include == TODO
2250  % \ifx\bbl@included@inis\@empty\else
2251  %   \bbl@replace\bbl@included@inis{ }{,}%
2252  %   \bbl@foreach\bbl@included@inis{%
2253  %     \openin\bbl@readstream=babel-##1.ini
2254  %     \bbl@extend@ini{#2}}%
2255  %   \closein\bbl@readstream
2256  % \fi
2257  % Post tasks
2258  % ----------
2259  % == subsequent calls after the first provide for a locale ==
2260  \ifx\bbl@inidata\@empty\else
2261    \bbl@extend@ini{#2}%
2262  \fi
2263  % == ensure captions ==
2264  \ifx\bbl@KVP@captions\@nnil\else
2265    \bbl@ifunset{bbl@extracaps@#2}%
2266      {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2267      {\bbl@exp{\\\babelensure[exclude=\\\today,
2268                include=\[bbl@extracaps@#2]]{#2}}}%
2269    \bbl@ifunset{bbl@ensure@\languagename}%
2270      {\bbl@exp{%
2271        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2272          \\\foreignlanguage{\languagename}%
2273          {####1}}}}%
2274      {}%
2275    \bbl@exp{%
2276      \\\bbl@toglobal\<bbl@ensure@\languagename>%
2277      \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2278  \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.

```
2279  \bbl@load@basic{#2}%
2280  % == script, language ==
2281  % Override the values from ini or defines them
2282  \ifx\bbl@KVP@script\@nnil\else
2283    \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2284  \fi
2285  \ifx\bbl@KVP@language\@nnil\else
2286    \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2287  \fi
2288  \ifcase\bbl@engine\or
2289    \bbl@ifunset{bbl@chrng@\languagename}{}%
```

```
2290      {\directlua{
2291        Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2292  \fi
2293  % == Line breaking: intraspace, intrapenalty ==
2294  % For CJK, East Asian, Southeast Asian, if interspace in ini
2295  \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2296    \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2297  \fi
2298  \bbl@provide@intraspace
2299  % == Line breaking: justification ==
2300  \ifx\bbl@KVP@justification\@nnil\else
2301      \let\bbl@KVP@linebreaking\bbl@KVP@justification
2302  \fi
2303  \ifx\bbl@KVP@linebreaking\@nnil\else
2304    \bbl@xin@{,\bbl@KVP@linebreaking,}%
2305      {,elongated,kashida,cjk,padding,unhyphenated,}%
2306    \ifin@
2307      \bbl@csarg\xdef
2308        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2309    \fi
2310  \fi
2311  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2312  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2313  \ifin@\bbl@arabicjust\fi
2314  % WIP
2315  \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2316  \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2317  % == Line breaking: hyphenate.other.(locale|script) ==
2318  \ifx\bbl@lbkflag\@empty
2319    \bbl@ifunset{bbl@hyotl@\languagename}{}%
2320      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2321        \bbl@startcommands*{\languagename}{}%
2322          \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2323            \ifcase\bbl@engine
2324              \ifnum##1<257
2325                \SetHyphenMap{\BabelLower{##1}{##1}}%
2326              \fi
2327            \else
2328              \SetHyphenMap{\BabelLower{##1}{##1}}%
2329            \fi}%
2330        \bbl@endcommands}%
2331    \bbl@ifunset{bbl@hyots@\languagename}{}%
2332      {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2333        \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2334          \ifcase\bbl@engine
2335            \ifnum##1<257
2336              \global\lccode##1=##1\relax
2337            \fi
2338          \else
2339            \global\lccode##1=##1\relax
2340          \fi}}%
2341  \fi
2342  % == Counters: maparabic ==
2343  % Native digits, if provided in ini (TeX level, xe and lua)
2344  \ifcase\bbl@engine\else
2345    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2346      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2347        \expandafter\expandafter\expandafter
2348        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2349        \ifx\bbl@KVP@maparabic\@nnil\else
2350          \ifx\bbl@latinarabic\@undefined
2351            \expandafter\let\expandafter\@arabic
2352              \csname bbl@counter@\languagename\endcsname
```

56

```
2353        \else    % ie, if layout=counters, which redefines \@arabic
2354          \expandafter\let\expandafter\bbl@latinarabic
2355            \csname bbl@counter@\languagename\endcsname
2356        \fi
2357      \fi
2358    \fi}%
2359  \fi
2360  % == Counters: mapdigits ==
2361  % > luababel.def
2362  % == Counters: alph, Alph ==
2363  \ifx\bbl@KVP@alph\@nnil\else
2364    \bbl@exp{%
2365      \\\bbl@add\<bbl@preextras@\languagename>{%
2366        \\\babel@save\\\@alph
2367        \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2368  \fi
2369  \ifx\bbl@KVP@Alph\@nnil\else
2370    \bbl@exp{%
2371      \\\bbl@add\<bbl@preextras@\languagename>{%
2372        \\\babel@save\\\@Alph
2373        \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2374  \fi
2375  % == Casing ==
2376  \bbl@release@casing
2377  \ifx\bbl@KVP@casing\@nnil\else
2378    \bbl@csarg\xdef{casing@\languagename}%
2379      {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2380  \fi
2381  % == Calendars ==
2382  \ifx\bbl@KVP@calendar\@nnil
2383    \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2384  \fi
2385  \def\bbl@tempe##1 ##2\@@{% Get first calendar
2386    \def\bbl@tempa{##1}}%
2387  \bbl@exp{\\\bbl@tempe\bbl@KVP@calendar\space\\\@@}%
2388  \def\bbl@tempe##1.##2.##3\@@{%
2389    \def\bbl@tempc{##1}%
2390    \def\bbl@tempb{##2}}%
2391  \expandafter\bbl@tempe\bbl@tempa..\@@
2392  \bbl@csarg\edef{calpr@\languagename}{%
2393    \ifx\bbl@tempc\@empty\else
2394      calendar=\bbl@tempc
2395    \fi
2396    \ifx\bbl@tempb\@empty\else
2397      ,variant=\bbl@tempb
2398    \fi}%
2399  % == engine specific extensions ==
2400  % Defined in XXXbabel.def
2401  \bbl@provide@extra{#2}%
2402  % == require.babel in ini ==
2403  % To load or reaload the babel-*.tex, if require.babel in ini
2404  \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
2405    \bbl@ifunset{bbl@rqtex@\languagename}{}%
2406      {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2407        \let\BabelBeforeIni\@gobbletwo
2408        \chardef\atcatcode=\catcode`\@
2409        \catcode`\@=11\relax
2410        \def\CurrentOption{#2}%
2411        \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2412        \catcode`\@=\atcatcode
2413        \let\atcatcode\relax
2414        \global\bbl@csarg\let{rqtex@\languagename}\relax
2415      \fi}%
```

```
2416    \bbl@foreach\bbl@calendars{%
2417      \bbl@ifunset{bbl@ca@##1}{%
2418        \chardef\atcatcode=\catcode`\@
2419        \catcode`\@=11\relax
2420        \InputIfFileExists{babel-ca-##1.tex}{}{}%
2421        \catcode`\@=\atcatcode
2422        \let\atcatcode\relax}%
2423      {}}%
2424  \fi
2425  % == frenchspacing ==
2426  \ifcase\bbl@howloaded\in@true\else\in@false\fi
2427  \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2428  \ifin@
2429    \bbl@extras@wrap{\\\bbl@pre@fs}%
2430      {\bbl@pre@fs}%
2431      {\bbl@post@fs}%
2432  \fi
2433  % == transforms ==
2434  % > luababel.def
2435  \def\CurrentOption{#2}%
2436  \@nameuse{bbl@icsave@#2}%
2437  % == main ==
2438  \ifx\bbl@KVP@main\@nnil  % Restore only if not 'main'
2439    \let\languagename\bbl@savelangname
2440    \chardef\localeid\bbl@savelocaleid\relax
2441  \fi
2442  % == hyphenrules (apply if current) ==
2443  \ifx\bbl@KVP@hyphenrules\@nnil\else
2444    \ifnum\bbl@savelocaleid=\localeid
2445      \language\@nameuse{l@\languagename}%
2446    \fi
2447  \fi}
```

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

```
2448 \def\bbl@provide@new#1{%
2449  \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2450  \@namedef{extras#1}{}%
2451  \@namedef{noextras#1}{}%
2452  \bbl@startcommands*{#1}{captions}%
2453    \ifx\bbl@KVP@captions\@nnil %     and also if import, implicit
2454      \def\bbl@tempb##1{%           elt for \bbl@captionslist
2455        \ifx##1\@nnil\else
2456          \bbl@exp{%
2457            \\\SetString\\##1{%
2458              \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2459          \expandafter\bbl@tempb
2460        \fi}%
2461      \expandafter\bbl@tempb\bbl@captionslist\@nnil
2462    \else
2463      \ifx\bbl@initoload\relax
2464        \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2465      \else
2466        \bbl@read@ini{\bbl@initoload}2%      % Same
2467      \fi
2468    \fi
2469  \StartBabelCommands*{#1}{date}%
2470    \ifx\bbl@KVP@date\@nnil
2471      \bbl@exp{%
2472        \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2473    \else
2474      \bbl@savetoday
2475      \bbl@savedate
```

```
2476        \fi
2477    \bbl@endcommands
2478    \bbl@load@basic{#1}%
2479    % == hyphenmins == (only if new)
2480    \bbl@exp{%
2481      \gdef\<#1hyphenmins>{%
2482        {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2483        {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2484    % == hyphenrules (also in renew) ==
2485    \bbl@provide@hyphens{#1}%
2486    \ifx\bbl@KVP@main\@nnil\else
2487      \expandafter\main@language\expandafter{#1}%
2488    \fi}
2489 %
2490 \def\bbl@provide@renew#1{%
2491    \ifx\bbl@KVP@captions\@nnil\else
2492      \StartBabelCommands*{#1}{captions}%
2493        \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2494      \EndBabelCommands
2495    \fi
2496    \ifx\bbl@KVP@date\@nnil\else
2497      \StartBabelCommands*{#1}{date}%
2498        \bbl@savetoday
2499        \bbl@savedate
2500      \EndBabelCommands
2501    \fi
2502    % == hyphenrules (also in new) ==
2503    \ifx\bbl@lbkflag\@empty
2504      \bbl@provide@hyphens{#1}%
2505    \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.

```
2506 \def\bbl@load@basic#1{%
2507    \ifcase\bbl@howloaded\or\or
2508      \ifcase\csname bbl@llevel@\languagename\endcsname
2509        \bbl@csarg\let{lname@\languagename}\relax
2510      \fi
2511    \fi
2512    \bbl@ifunset{bbl@lname@#1}%
2513      {\def\BabelBeforeIni##1##2{%
2514        \begingroup
2515          \let\bbl@ini@captions@aux\@gobbletwo
2516          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2517          \bbl@read@ini{##1}1%
2518          \ifx\bbl@initoload\relax\endinput\fi
2519        \endgroup}%
2520      \begingroup       % boxed, to avoid extra spaces:
2521        \ifx\bbl@initoload\relax
2522          \bbl@input@texini{#1}%
2523        \else
2524          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2525        \fi
2526      \endgroup}%
2527      {}}
```

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.

```
2528 \def\bbl@provide@hyphens#1{%
2529    \@tempcnta\m@ne  % a flag
2530    \ifx\bbl@KVP@hyphenrules\@nnil\else
2531      \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2532      \bbl@foreach\bbl@KVP@hyphenrules{%
```

```
2533      \ifnum\@tempcnta=\m@ne    % if not yet found
2534        \bbl@ifsamestring{##1}{+}%
2535          {\bbl@carg\addlanguage{l@##1}}%
2536          {}%
2537        \bbl@ifunset{l@##1}% After a possible +
2538          {}%
2539          {\@tempcnta\@nameuse{l@##1}}%
2540      \fi}%
2541    \ifnum\@tempcnta=\m@ne
2542      \bbl@warning{%
2543        Requested 'hyphenrules' for '\languagename' not found:\\%
2544        \bbl@KVP@hyphenrules.\\%
2545        Using the default value. Reported}%
2546    \fi
2547  \fi
2548  \ifnum\@tempcnta=\m@ne            % if no opt or no language in opt found
2549    \ifx\bbl@KVP@captions@@\@nnil % TODO. Hackish. See above.
2550      \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2551        {\bbl@exp{\\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2552          {}%
2553          {\bbl@ifunset{l@\bbl@cl{hyphr}}%
2554            {}%                  if hyphenrules found:
2555            {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2556    \fi
2557  \fi
2558  \bbl@ifunset{l@#1}%
2559    {\ifnum\@tempcnta=\m@ne
2560        \bbl@carg\adddialect{l@#1}\language
2561     \else
2562        \bbl@carg\adddialect{l@#1}\@tempcnta
2563     \fi}%
2564    {\ifnum\@tempcnta=\m@ne\else
2565        \global\bbl@carg\chardef{l@#1}\@tempcnta
2566     \fi}}
```

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

```
2567 \def\bbl@input@texini#1{%
2568   \bbl@bsphack
2569     \bbl@exp{%
2570       \catcode`\\\%=14 \catcode`\\\\=0
2571       \catcode`\\\{=1  \catcode`\\\}=2
2572       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2573       \catcode`\\\%=\the\catcode`\%\relax
2574       \catcode`\\\\=\the\catcode`\\\relax
2575       \catcode`\\\{=\the\catcode`\{\relax
2576       \catcode`\\\}=\the\catcode`\}\relax}%
2577   \bbl@esphack}
```

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

```
2578 \def\bbl@iniline#1\bbl@iniline{%
2579   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2580 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2581 \def\bbl@iniskip#1\@@{}%     if starts with ;
2582 \def\bbl@inistore#1=#2\@@{%     full (default)
2583   \bbl@trim@def\bbl@tempa{#1}%
2584   \bbl@trim\toks@{#2}%
2585   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2586   \ifin@\else
2587     \bbl@xin@{,identification/include.}%
2588             {,\bbl@section/\bbl@tempa}%
2589     \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
```

```
2590    \bbl@exp{%
2591      \\\g@addto@macro\\\bbl@inidata{%
2592        \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2593    \fi}
2594 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2595    \bbl@trim@def\bbl@tempa{#1}%
2596    \bbl@trim\toks@{#2}%
2597    \bbl@xin@{.identification.}{.\bbl@section.}%
2598    \ifin@
2599      \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2600        \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2601    \fi}
```

4.19. Main loop in 'provide'

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

```
2602 \def\bbl@loop@ini{%
2603    \loop
2604      \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2605        \endlinechar\m@ne
2606        \read\bbl@readstream to \bbl@line
2607        \endlinechar`\^^M
2608        \ifx\bbl@line\@empty\else
2609          \expandafter\bbl@iniline\bbl@line\bbl@iniline
2610        \fi
2611    \repeat}
2612 \ifx\bbl@readstream\@undefined
2613    \csname newread\endcsname\bbl@readstream
2614 \fi
2615 \def\bbl@read@ini#1#2{%
2616    \global\let\bbl@extend@ini\@gobble
2617    \openin\bbl@readstream=babel-#1.ini
2618    \ifeof\bbl@readstream
2619      \bbl@error{no-ini-file}{#1}{}{}%
2620    \else
2621      % == Store ini data in \bbl@inidata ==
2622      \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2623      \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2624      \bbl@info{Importing
2625                   \ifcase#2font and identification \or basic \fi
2626                    data for \languagename\\%
2627                  from babel-#1.ini. Reported}%
2628      \ifnum#2=\z@
2629        \global\let\bbl@inidata\@empty
2630        \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2631      \fi
2632      \def\bbl@section{identification}%
2633      \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2634      \bbl@inistore load.level=#2\@@
2635      \bbl@loop@ini
2636      % == Process stored data ==
2637      \bbl@csarg\xdef{lini@\languagename}{#1}%
2638      \bbl@read@ini@aux
2639      % == 'Export' data ==
2640      \bbl@ini@exports{#2}%
2641      \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2642      \global\let\bbl@inidata\@empty
2643      \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
```

61

```
2644     \bbl@toglobal\bbl@ini@loaded
2645   \fi
2646   \closein\bbl@readstream}
2647 \def\bbl@read@ini@aux{%
2648   \let\bbl@savestrings\@empty
2649   \let\bbl@savetoday\@empty
2650   \let\bbl@savedate\@empty
2651   \def\bbl@elt##1##2##3{%
2652     \def\bbl@section{##1}%
2653     \in@{=date.}{=##1}% Find a better place
2654     \ifin@
2655       \bbl@ifunset{bbl@inikv@##1}%
2656         {\bbl@ini@calendar{##1}}%
2657         {}%
2658     \fi
2659     \bbl@ifunset{bbl@inikv@##1}{}%
2660       {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2661   \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.

```
2662 \def\bbl@extend@ini@aux#1{%
2663   \bbl@startcommands*{#1}{captions}%
2664     % Activate captions/... and modify exports
2665     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2666       \setlocalecaption{#1}{##1}{##2}}%
2667     \def\bbl@inikv@captions##1##2{%
2668       \bbl@ini@captions@aux{##1}{##2}}%
2669     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2670     \def\bbl@exportkey##1##2##3{%
2671       \bbl@ifunset{bbl@@kv@##2}{}%
2672         {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
2673           \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
2674         \fi}}%
2675     % As with \bbl@read@ini, but with some changes
2676     \bbl@read@ini@aux
2677     \bbl@ini@exports\tw@
2678     % Update inidata@lang by pretending the ini is read.
2679     \def\bbl@elt##1##2##3{%
2680       \def\bbl@section{##1}%
2681       \bbl@iniline##2=##3\bbl@iniline}%
2682     \csname bbl@inidata@#1\endcsname
2683     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2684   \StartBabelCommands*{#1}{date}% And from the import stuff
2685     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2686     \bbl@savetoday
2687     \bbl@savedate
2688   \bbl@endcommands}
```

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

```
2689 \def\bbl@ini@calendar#1{%
2690   \lowercase{\def\bbl@tempa{=#1=}}%
2691   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2692   \bbl@replace\bbl@tempa{=date.}{}%
2693   \in@{.licr=}{#1=}%
2694   \ifin@
2695     \ifcase\bbl@engine
2696       \bbl@replace\bbl@tempa{.licr=}{}%
2697     \else
2698       \let\bbl@tempa\relax
2699     \fi
2700   \fi
2701   \ifx\bbl@tempa\relax\else
2702     \bbl@replace\bbl@tempa{=}{}%
```

```
2703    \ifx\bbl@tempa\@empty\else
2704      \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2705    \fi
2706    \bbl@exp{%
2707      \def\<bbl@inikv@#1>####1####2{%
2708        \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2709  \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).

```
2710 \def\bbl@renewinikey#1/#2\@@#3{%
2711    \edef\bbl@tempa{\zap@space #1 \@empty}%    section
2712    \edef\bbl@tempb{\zap@space #2 \@empty}%    key
2713    \bbl@trim\toks@{#3}%                       value
2714    \bbl@exp{%
2715      \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2716      \\\g@addto@macro\\\bbl@inidata{%
2717        \\\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.

```
2718 \def\bbl@exportkey#1#2#3{%
2719    \bbl@ifunset{bbl@@kv@#2}%
2720      {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2721      {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2722        \bbl@csarg\gdef{#1@\languagename}{#3}%
2723       \else
2724        \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2725      \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.

```
2726 \def\bbl@iniwarning#1{%
2727    \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2728      {\bbl@warning{%
2729        From babel-\bbl@cs{lini@\languagename}.ini:\\%
2730        \bbl@cs{@kv@identification.warning#1}\\%
2731        Reported }}}
2732 %
2733 \let\bbl@release@transforms\@empty
2734 \let\bbl@release@casing\@empty
2735 \def\bbl@ini@exports#1{%
2736    % Identification always exported
2737    \bbl@iniwarning{}%
2738    \ifcase\bbl@engine
2739      \bbl@iniwarning{.pdflatex}%
2740    \or
2741      \bbl@iniwarning{.lualatex}%
2742    \or
2743      \bbl@iniwarning{.xelatex}%
2744    \fi%
2745    \bbl@exportkey{llevel}{identification.load.level}{}%
2746    \bbl@exportkey{elname}{identification.name.english}{}%
2747    \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
2748      {\csname bbl@elname@\languagename\endcsname}}%
2749    \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2750    % Somewhat hackish. TODO:
```

```
2751  \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2752  \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2753  \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2754  \bbl@exportkey{esname}{identification.script.name}{}%
2755  \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
2756    {\csname bbl@esname@\languagename\endcsname}}%
2757  \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
2758  \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2759  \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2760  \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2761  \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2762  \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2763  \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2764  % Also maps bcp47 -> languagename
2765  \ifbbl@bcptoname
2766    \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
2767  \fi
2768  \ifcase\bbl@engine\or
2769    \directlua{%
2770      Babel.locale_props[\the\bbl@cs{id@@\languagename}].script
2771        = '\bbl@cl{sbcp}'}%
2772  \fi
2773  % Conditional
2774  \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
2775    \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2776    \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2777    \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2778    \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2779    \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2780    \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2781    \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2782    \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2783    \bbl@exportkey{intsp}{typography.intraspace}{}%
2784    \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2785    \bbl@exportkey{chrng}{characters.ranges}{}%
2786    \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2787    \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2788    \ifnum#1=\tw@          % only (re)new
2789      \bbl@exportkey{rqtex}{identification.require.babel}{}%
2790      \bbl@toglobal\bbl@savetoday
2791      \bbl@toglobal\bbl@savedate
2792      \bbl@savestrings
2793    \fi
2794  \fi}
```

4.20. Processing keys in `ini`

A shared handler for key=val lines to be stored in \bbl@@kv@⟨*section*⟩.⟨*key*⟩.

```
2795  \def\bbl@inikv#1#2{%      key=value
2796    \toks@{#2}%             This hides #'s from ini values
2797    \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
2798  \let\bbl@inikv@identification\bbl@inikv
2799  \let\bbl@inikv@date\bbl@inikv
2800  \let\bbl@inikv@typography\bbl@inikv
2801  \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.

```
2802  \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2803  \def\bbl@inikv@characters#1#2{%
```

```
2804  \bbl@ifsamestring{#1}{casing}%  eg, casing = uV
2805    {\bbl@exp{%
2806       \\\g@addto@macro\\\bbl@release@casing{%
2807         \\\bbl@casemapping{}{\languagename}{\unexpanded{#2}}}}}%
2808    {\in@{$casing.}{$#1}%  eg, casing.Uv = uV
2809     \ifin@
2810       \lowercase{\def\bbl@tempb{#1}}%
2811       \bbl@replace\bbl@tempb{casing.}{}%
2812       \bbl@exp{\\\g@addto@macro\\\bbl@release@casing{%
2813         \\\bbl@casemapping
2814           {\\\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}}%
2815     \else
2816       \bbl@inikv{#1}{#2}%
2817     \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'.

```
2818 \def\bbl@inikv@counters#1#2{%
2819  \bbl@ifsamestring{#1}{digits}%
2820    {\bbl@error{digits-is-reserved}{}{}{}}%
2821    {}%
2822  \def\bbl@tempc{#1}%
2823  \bbl@trim@def{\bbl@tempb*}{#2}%
2824  \in@{.1$}{#1$}%
2825  \ifin@
2826    \bbl@replace\bbl@tempc{.1}{}%
2827    \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
2828      \noexpand\bbl@alphnumeral{\bbl@tempc}}%
2829  \fi
2830  \in@{.F.}{#1}%
2831  \ifin@\else\in@{.S.}{#1}\fi
2832  \ifin@
2833    \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
2834  \else
2835    \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2836    \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
2837    \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
2838  \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.

```
2839 \ifcase\bbl@engine
2840  \bbl@csarg\def{inikv@captions.licr}#1#2{%
2841    \bbl@ini@captions@aux{#1}{#2}}
2842 \else
2843  \def\bbl@inikv@captions#1#2{%
2844    \bbl@ini@captions@aux{#1}{#2}}
2845 \fi
```

The auxiliary macro for captions define \⟨caption⟩name.

```
2846 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2847  \bbl@replace\bbl@tempa{.template}{}%
2848  \def\bbl@toreplace{#1{}}%
2849  \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2850  \bbl@replace\bbl@toreplace{[[}{\csname}%
2851  \bbl@replace\bbl@toreplace{[}{\csname the}%
2852  \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
2853  \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
2854  \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2855  \ifin@
2856    \@nameuse{bbl@patch\bbl@tempa}%
2857    \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
```

```
2858    \fi
2859    \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2860    \ifin@
2861      \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2862      \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2863        \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\\languagename}%
2864          {\[fnum@\bbl@tempa]}%
2865          {\\\@nameuse{bbl@\bbl@tempa fmt@\\\languagename}}}}%
2866    \fi}
2867 \def\bbl@ini@captions@aux#1#2{%
2868    \bbl@trim@def\bbl@tempa{#1}%
2869    \bbl@xin@{.template}{\bbl@tempa}%
2870    \ifin@
2871      \bbl@ini@captions@template{#2}\languagename
2872    \else
2873      \bbl@ifblank{#2}%
2874        {\bbl@exp{%
2875          \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
2876        {\bbl@trim\toks@{#2}}%
2877      \bbl@exp{%
2878        \\\bbl@add\\\bbl@savestrings{%
2879          \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
2880      \toks@\expandafter{\bbl@captionslist}%
2881      \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
2882      \ifin@\else
2883        \bbl@exp{%
2884          \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
2885          \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
2886      \fi
2887    \fi}
```

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

```
2888 \def\bbl@list@the{%
2889    part,chapter,section,subsection,subsubsection,paragraph,%
2890    subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2891    table,page,footnote,mpfootnote,mpfn}
2892 \def\bbl@map@cnt#1{%   #1:roman,etc, // #2:enumi,etc
2893    \bbl@ifunset{bbl@map@#1@\languagename}%
2894      {\@nameuse{#1}}%
2895      {\@nameuse{bbl@map@#1@\languagename}}}
2896 \def\bbl@inikv@labels#1#2{%
2897    \in@{.map}{#1}%
2898    \ifin@
2899      \ifx\bbl@KVP@labels\@nnil\else
2900        \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
2901        \ifin@
2902          \def\bbl@tempc{#1}%
2903          \bbl@replace\bbl@tempc{.map}{}%
2904          \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
2905          \bbl@exp{%
2906            \gdef\<bbl@map@\bbl@tempc @\languagename>%
2907              {\ifin@\<#2>\else\\\localecounter{#2}\fi}}%
2908          \bbl@foreach\bbl@list@the{%
2909            \bbl@ifunset{the##1}{}%
2910              {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
2911               \bbl@exp{%
2912                 \\\bbl@sreplace\<the##1>%
2913                   {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
2914                 \\\bbl@sreplace\<the##1>%
2915                   {\<\@empty @\bbl@tempc>\<\c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2916               \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2917                 \toks@\expandafter\expandafter\expandafter{%
2918                   \csname the##1\endcsname}%
```

```
2919              \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
2920            \fi}}%
2921      \fi
2922    \fi
2923  %
2924  \else
2925    %
2926    % The following code is still under study. You can test it and make
2927    % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
2928    % language dependent.
2929    \in@{enumerate.}{#1}%
2930    \ifin@
2931      \def\bbl@tempa{#1}%
2932      \bbl@replace\bbl@tempa{enumerate.}{}%
2933      \def\bbl@toreplace{#2}%
2934      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2935      \bbl@replace\bbl@toreplace{[}{\csname the}%
2936      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
2937      \toks@\expandafter{\bbl@toreplace}%
2938      % TODO. Execute only once:
2939      \bbl@exp{%
2940        \\\bbl@add\<extras\languagename>{%
2941          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
2942          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
2943        \\\bbl@toglobal\<extras\languagename>}%
2944    \fi
2945  \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.

```
2946 \def\bbl@chaptype{chapter}
2947 \ifx\@makechapterhead\@undefined
2948   \let\bbl@patchchapter\relax
2949 \else\ifx\thechapter\@undefined
2950   \let\bbl@patchchapter\relax
2951 \else\ifx\ps@headings\@undefined
2952   \let\bbl@patchchapter\relax
2953 \else
2954   \def\bbl@patchchapter{%
2955     \global\let\bbl@patchchapter\relax
2956     \gdef\bbl@chfmt{%
2957       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
2958         {\@chapapp\space\thechapter}%
2959         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}%
2960     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
2961     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
2962     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
2963     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
2964     \bbl@toglobal\appendix
2965     \bbl@toglobal\ps@headings
2966     \bbl@toglobal\chaptermark
2967     \bbl@toglobal\@makechapterhead}
2968   \let\bbl@patchappendix\bbl@patchchapter
2969 \fi\fi\fi
2970 \ifx\@part\@undefined
2971   \let\bbl@patchpart\relax
2972 \else
2973   \def\bbl@patchpart{%
2974     \global\let\bbl@patchpart\relax
2975     \gdef\bbl@partformat{%
2976       \bbl@ifunset{bbl@partfmt@\languagename}%
```

```
2977        {\partname\nobreakspace\thepart}
2978        {\@nameuse{bbl@partfmt@\languagename}}}
2979    \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
2980    \bbl@toglobal\@part}
2981 \fi
```

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

```
2982 \let\bbl@calendar\@empty
2983 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
2984 \def\bbl@localedate#1#2#3#4{%
2985    \begingroup
2986      \edef\bbl@they{#2}%
2987      \edef\bbl@them{#3}%
2988      \edef\bbl@thed{#4}%
2989      \edef\bbl@tempe{%
2990        \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},%
2991        #1}%
2992      \bbl@replace\bbl@tempe{ }{}%
2993      \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
2994      \bbl@replace\bbl@tempe{convert}{convert=}%
2995      \let\bbl@ld@calendar\@empty
2996      \let\bbl@ld@variant\@empty
2997      \let\bbl@ld@convert\relax
2998      \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
2999      \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3000      \bbl@replace\bbl@ld@calendar{gregorian}{}%
3001      \ifx\bbl@ld@calendar\@empty\else
3002        \ifx\bbl@ld@convert\relax\else
3003          \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3004            {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3005        \fi
3006      \fi
3007      \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3008      \edef\bbl@calendar{% Used in \month..., too
3009        \bbl@ld@calendar
3010        \ifx\bbl@ld@variant\@empty\else
3011          .\bbl@ld@variant
3012        \fi}%
3013      \bbl@cased
3014        {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3015          \bbl@they\bbl@them\bbl@thed}%
3016    \endgroup}
3017 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3018 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3019    \bbl@trim@def\bbl@tempa{#1.#2}%
3020    \bbl@ifsamestring{\bbl@tempa}{months.wide}%       to savedate
3021      {\bbl@trim@def\bbl@tempa{#3}%
3022      \bbl@trim\toks@{#5}%
3023      \@temptokena\expandafter{\bbl@savedate}%
3024      \bbl@exp{%   Reverse order - in ini last wins
3025        \def\\\bbl@savedate{%
3026          \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3027          \the\@temptokena}}%
3028      {\bbl@ifsamestring{\bbl@tempa}{date.long}%     defined now
3029        {\lowercase{\def\bbl@tempb{#6}}%
3030        \bbl@trim@def\bbl@toreplace{#5}%
3031        \bbl@TG@@date
3032        \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3033        \ifx\bbl@savetoday\@empty
3034          \bbl@exp{% TODO. Move to a better place.
3035            \\\AfterBabelCommands{%
3036              \gdef\<\languagename date>{\\\protect\<\languagename date >}%
```

```
3037            \gdef\<\languagename date >{\\\bbl@printdate{\languagename}}}%
3038          \def\\\bbl@savetoday{%
3039            \\\SetString\\\today{%
3040              \<\languagename date>[convert]%
3041                {\\\the\year}{\\\the\month}{\\\the\day}}}}%
3042        \fi}%
3043      {}}}
3044 \def\bbl@printdate#1{%
3045   \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}}
3046 \def\bbl@printdate@i#1[#2]#3#4#5{%
3047   \bbl@usedategrouptrue
3048   \@nameuse{bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}}
```

4.21. French spacing (again)

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

```
3049 \AddToHook{begindocument/before}{%
3050   \let\bbl@normalsf\normalsfcodes
3051   \let\normalsfcodes\relax}
3052 \AtBeginDocument{%
3053   \ifx\bbl@normalsf\@empty
3054     \ifnum\sfcode`\.=\@m
3055       \let\normalsfcodes\frenchspacing
3056     \else
3057       \let\normalsfcodes\nonfrenchspacing
3058     \fi
3059   \else
3060     \let\normalsfcodes\bbl@normalsf
3061   \fi}
```

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).

```
3062 \let\bbl@calendar\@empty
3063 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3064   \@nameuse{bbl@ca@#2}#1\@@}
3065 \newcommand\BabelDateSpace{\nobreakspace}
3066 \newcommand\BabelDateDot{.\@}   % TODO. \let instead of repeating
3067 \newcommand\BabelDated[1]{{\number#1}}
3068 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3069 \newcommand\BabelDateM[1]{{\number#1}}
3070 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3071 \newcommand\BabelDateMMMM[1]{{%
3072   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3073 \newcommand\BabelDatey[1]{{\number#1}}%
3074 \newcommand\BabelDateyy[1]{{%
3075   \ifnum#1<10 0\number#1 %
3076   \else\ifnum#1<100 \number#1 %
3077   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3078   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3079   \else
3080     \bbl@error{limit-two-digits}{}{}{}%
3081   \fi\fi\fi\fi}}
3082 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3083 \newcommand\BabelDateU[1]{{\number#1}}%
3084 \def\bbl@replace@finish@iii#1{%
3085   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3086 \def\bbl@TG@@date{%
3087   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3088   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
```

```
3089  \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3090  \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3091  \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3092  \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3093  \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3094  \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3095  \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3096  \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3097  \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}%
3098  \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecntr[####1|]}%
3099  \bbl@replace\bbl@toreplace{[U|]}{\bbl@datecntr[####1|]}%
3100  \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecntr[####2|]}%
3101  \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecntr[####3|]}%
3102  \bbl@replace@finish@iii\bbl@toreplace}
3103 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3104 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3105 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3106 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3107 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3108   #1[#2]{#3}{#4}{#5}}
3109 \begingroup %  A hack. TODO. Don't require a specific order
3110   \catcode`\%=12
3111   \catcode`\&=14
3112   \gdef\bbl@transforms#1#2#3{&%
3113     \directlua{
3114       local str = [==[#2]==]
3115       str = str:gsub('%.%d+%.%d+$', '')
3116       token.set_macro('babeltempa', str)
3117     }&%
3118     \def\babeltempc{}&%
3119     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3120     \ifin@\else
3121       \bbl@xin@{:\babeltempa,}{,\bbl@KVP@transforms,}&%
3122     \fi
3123     \ifin@
3124       \bbl@foreach\bbl@KVP@transforms{&%
3125         \bbl@xin@{:\babeltempa,}{,##1,}&%
3126         \ifin@  &% font:font:transform syntax
3127           \directlua{
3128             local t = {}
3129             for m in string.gmatch('##1'..':', '(.-):') do
3130               table.insert(t, m)
3131             end
3132             table.remove(t)
3133             token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3134           }&%
3135         \fi}&%
3136       \in@{.0$}{#2$}&%
3137       \ifin@
3138         \directlua{&% (attribute) syntax
3139           local str = string.match([[\bbl@KVP@transforms]],
3140                       '%(([^%(]-)%)[^%)]-\babeltempa')
3141           if str == nil then
3142             token.set_macro('babeltempb', '')
3143           else
3144             token.set_macro('babeltempb', ',attribute=' .. str)
3145           end
3146         }&%
3147       \toks@{#3}&%
3148       \bbl@exp{&%
3149         \\\g@addto@macro\\\bbl@release@transforms{&%
```

70

```
3150          \relax  &% Closes previous \bbl@transforms@aux
3151            \\\bbl@transforms@aux
3152              \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3153                {\languagename}{\the\toks@}}}&%
3154        \else
3155          \g@addto@macro\bbl@release@transforms{, {#3}}&%
3156        \fi
3157      \fi}
3158 \endgroup
```

4.22. Handle language system

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

```
3159 \def\bbl@provide@lsys#1{%
3160   \bbl@ifunset{bbl@lname@#1}%
3161     {\bbl@load@info{#1}}%
3162     {}%
3163   \bbl@csarg\let{lsys@#1}\@empty
3164   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3165   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3166   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3167   \bbl@ifunset{bbl@lname@#1}{}%
3168     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3169   \ifcase\bbl@engine\or\or
3170     \bbl@ifunset{bbl@prehc@#1}{}%
3171       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3172         {}%
3173         {\ifx\bbl@xenohyph\@undefined
3174           \global\let\bbl@xenohyph\bbl@xenohyph@d
3175           \ifx\AtBeginDocument\@notprerr
3176             \expandafter\@secondoftwo  % to execute right now
3177           \fi
3178           \AtBeginDocument{%
3179             \bbl@patchfont{\bbl@xenohyph}%
3180             {\expandafter\select@language\expandafter{\languagename}}}%
3181       \fi}}%
3182   \fi
3183   \bbl@csarg\bbl@toglobal{lsys@#1}}
3184 \def\bbl@xenohyph@d{%
3185   \bbl@ifset{bbl@prehc@\languagename}%
3186     {\ifnum\hyphenchar\font=\defaulthyphenchar
3187        \iffontchar\font\bbl@cl{prehc}\relax
3188          \hyphenchar\font\bbl@cl{prehc}\relax
3189        \else\iffontchar\font"200B
3190          \hyphenchar\font"200B
3191        \else
3192          \bbl@warning
3193            {Neither 0 nor ZERO WIDTH SPACE are available\\%
3194             in the current font, and therefore the hyphen\\%
3195             will be printed. Try changing the fontspec's\\%
3196             'HyphenChar' to another value, but be aware\\%
3197             this setting is not safe (see the manual).\\%
3198             Reported}%
3199          \hyphenchar\font\defaulthyphenchar
3200        \fi\fi
3201     \fi}%
3202     {\hyphenchar\font\defaulthyphenchar}}
3203 % \fi}
```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly,

but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```
3204 \def\bbl@load@info#1{%
3205   \def\BabelBeforeIni##1##2{%
3206     \begingroup
3207       \bbl@read@ini{##1}0%
3208       \endinput          % babel- .tex may contain onlypreamble's
3209     \endgroup}%               boxed, to avoid extra spaces:
3210   {\bbl@input@texini{#1}}}
```

4.23. Numerals

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 TeX. Non-digits characters are kept. The first macro is the generic "localized" command.

```
3211 \def\bbl@setdigits#1#2#3#4#5{%
3212   \bbl@exp{%
3213     \def\<\languagename digits>####1{%        ie, \langdigits
3214       \<bbl@digits@\languagename>####1\\\@nil}%
3215     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3216     \def\<\languagename counter>####1{%       ie, \langcounter
3217       \\\expandafter\<bbl@counter@\languagename>%
3218       \\\csname c@####1\endcsname}%
3219     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3220       \\\expandafter\<bbl@digits@\languagename>%
3221       \\\number####1\\\@nil}}%
3222   \def\bbl@tempa##1##2##3##4##5{%
3223     \bbl@exp{%    Wow, quite a lot of hashes! :-(
3224       \def\<bbl@digits@\languagename>########1{%
3225         \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
3226         \\\else
3227           \\\ifx0########1#1%
3228           \\\else\\\ifx1########1#2%
3229           \\\else\\\ifx2########1#3%
3230           \\\else\\\ifx3########1#4%
3231           \\\else\\\ifx4########1#5%
3232           \\\else\\\ifx5########1##1%
3233           \\\else\\\ifx6########1##2%
3234           \\\else\\\ifx7########1##3%
3235           \\\else\\\ifx8########1##4%
3236           \\\else\\\ifx9########1##5%
3237           \\\else########1%
3238           \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3239           \\\expandafter\<bbl@digits@\languagename>%
3240         \\\fi}}}%
3241   \bbl@tempa}
```

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

```
3242 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3243   \ifx\\#1%              % \\ before, in case #1 is multiletter
3244     \bbl@exp{%
3245       \def\\\bbl@tempa####1{%
3246         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3247   \else
3248     \toks@\expandafter{\the\toks@\or #1}%
3249     \expandafter\bbl@buildifcase
3250   \fi}
```

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

```
3251 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3252 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3253 \newcommand\localecounter[2]{%
3254   \expandafter\bbl@localecntr
3255   \expandafter{\number\csname c@#2\endcsname}{#1}}
3256 \def\bbl@alphnumeral#1#2{%
3257   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3258 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3259   \ifcase\@car#8\@nil\or   % Currently <10000, but prepared for bigger
3260     \bbl@alphnumeral@ii{#9}000000#1\or
3261     \bbl@alphnumeral@ii{#9}00000#1#2\or
3262     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3263     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3264     \bbl@alphnum@invalid{>9999}%
3265   \fi}
3266 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3267   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3268     {\bbl@cs{cntr@#1.4@\languagename}#5%
3269      \bbl@cs{cntr@#1.3@\languagename}#6%
3270      \bbl@cs{cntr@#1.2@\languagename}#7%
3271      \bbl@cs{cntr@#1.1@\languagename}#8%
3272      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3273        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3274          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3275      \fi}%
3276     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3277 \def\bbl@alphnum@invalid#1{%
3278   \bbl@error{alphabetic-too-large}{#1}{}{}}
```

4.24. Casing

```
3279 \newcommand\BabelUppercaseMapping[3]{%
3280   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3281 \newcommand\BabelTitlecaseMapping[3]{%
3282   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3283 \newcommand\BabelLowercaseMapping[3]{%
3284   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
```

  The parser for casing and casing.⟨*variant*⟩.

```
3285 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3286   \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3287 \else
3288   \def\bbl@utftocode#1{\expandafter`\string#1}
3289 \fi
3290 \def\bbl@casemapping#1#2#3{% 1:variant
3291   \def\bbl@tempa##1 ##2{% Loop
3292     \bbl@casemapping@i{##1}%
3293     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3294   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}%  Language code
3295   \def\bbl@tempe{0}%   Mode (upper/lower...)
3296   \def\bbl@tempc{#3 }% Casing list
3297   \expandafter\bbl@tempa\bbl@tempc\@empty}
3298 \def\bbl@casemapping@i#1{%
3299   \def\bbl@tempb{#1}%
3300   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3301     \@nameuse{regex_replace_all:nnN}%
3302       {[\x{c0}-\x{ff}][\x{80}-\x{bf}]*}{{\0}}\bbl@tempb
3303   \else
3304     \@nameuse{regex_replace_all:nnN}{.}{{\0}}\bbl@tempb % TODO. needed?
3305   \fi
3306   \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3307 \def\bbl@casemapping@ii#1#2#3\@@{%
3308   \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3309   \ifin@
```

73

```
3310      \edef\bbl@tempe{%
3311         \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3312      \else
3313        \ifcase\bbl@tempe\relax
3314           \DeclareUppercaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
3315           \DeclareLowercaseMapping[\bbl@templ]{\bbl@utftocode{#2}}{#1}%
3316        \or
3317           \DeclareUppercaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
3318        \or
3319           \DeclareLowercaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
3320        \or
3321           \DeclareTitlecaseMapping[\bbl@templ]{\bbl@utftocode{#1}}{#2}%
3322        \fi
3323      \fi}
```

4.25. Getting info

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

```
3324 \def\bbl@localeinfo#1#2{%
3325   \bbl@ifunset{bbl@info@#2}{#1}%
3326     {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3327        {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3328 \newcommand\localeinfo[1]{%
3329   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3330     \bbl@afterelse\bbl@localeinfo{}%
3331   \else
3332     \bbl@localeinfo
3333        {\bbl@error{no-ini-info}{}{}{}}%
3334        {#1}%
3335   \fi}
3336 % \@namedef{bbl@info@name.locale}{lcname}
3337 \@namedef{bbl@info@tag.ini}{lini}
3338 \@namedef{bbl@info@name.english}{elname}
3339 \@namedef{bbl@info@name.opentype}{lname}
3340 \@namedef{bbl@info@tag.bcp47}{tbcp}
3341 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3342 \@namedef{bbl@info@tag.opentype}{lotf}
3343 \@namedef{bbl@info@script.name}{esname}
3344 \@namedef{bbl@info@script.name.opentype}{sname}
3345 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3346 \@namedef{bbl@info@script.tag.opentype}{sotf}
3347 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3348 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3349 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3350 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3351 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}
```

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

```
3352 ⟨⟨*More package options⟩⟩ ≡
3353 \DeclareOption{ensureinfo=off}{}
3354 ⟨⟨/More package options⟩⟩
3355 \let\bbl@ensureinfo\@gobble
3356 \newcommand\BabelEnsureInfo{%
3357   \ifx\InputIfFileExists\@undefined\else
3358     \def\bbl@ensureinfo##1{%
3359        \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3360   \fi
3361   \bbl@foreach\bbl@loaded{{%
3362     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3363     \def\languagename{##1}%
3364     \bbl@ensureinfo{##1}}}}
3365 \@ifpackagewith{babel}{ensureinfo=off}{}
3366   {\AtEndOfPackage{% Test for plain.
```

```
3367        \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}
```

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

```
3368 \newcommand\getlocaleproperty{%
3369    \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3370 \def\bbl@getproperty@s#1#2#3{%
3371    \let#1\relax
3372    \def\bbl@elt##1##2##3{%
3373      \bbl@ifsamestring{##1/##2}{#3}%
3374        {\providecommand#1{##3}%
3375         \def\bbl@elt####1####2####3{}}%
3376        {}}%
3377    \bbl@cs{inidata@#2}}%
3378 \def\bbl@getproperty@x#1#2#3{%
3379    \bbl@getproperty@s{#1}{#2}{#3}%
3380    \ifx#1\relax
3381      \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3382    \fi}
3383 \let\bbl@ini@loaded\@empty
3384 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3385 \def\ShowLocaleProperties#1{%
3386    \typeout{}%
3387    \typeout{*** Properties for language '#1' ***}
3388    \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3389    \@nameuse{bbl@inidata@#1}%
3390    \typeout{*******}}
```

4.26. BCP-47 related commands

```
3391 \newif\ifbbl@bcpallowed
3392 \bbl@bcpallowedfalse
3393 \def\bbl@provide@locale{%
3394    \ifx\babelprovide\@undefined
3395      \bbl@error{base-on-the-fly}{}{}{}%
3396    \fi
3397    \let\bbl@auxname\languagename % Still necessary. %^^A TODO
3398    \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
3399      {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
3400    \ifbbl@bcpallowed
3401      \expandafter\ifx\csname date\languagename\endcsname\relax
3402        \expandafter
3403        \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
3404        \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
3405          \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
3406          \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
3407          \expandafter\ifx\csname date\languagename\endcsname\relax
3408            \let\bbl@initoload\bbl@bcp
3409            \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
3410            \let\bbl@initoload\relax
3411          \fi
3412          \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
3413        \fi
3414      \fi
3415    \fi
3416    \expandafter\ifx\csname date\languagename\endcsname\relax
3417      \IfFileExists{babel-\languagename.tex}%
3418        {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
3419        {}%
3420    \fi}
```

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

change.

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

```
3421 \providecommand\BCPdata{}
3422 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3423   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3424   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3425     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3426       {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3427       {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3428   \def\bbl@bcpdata@ii#1#2{%
3429     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3430       {\bbl@error{unknown-ini-field}{#1}{}{}}%
3431       {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3432         {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}}
3433 \fi
3434 \@namedef{bbl@info@casing.tag.bcp47}{casing}
```

5. Adjusting the Babel behavior

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

```
3435 \newcommand\babeladjust[1]{%  TODO. Error handling.
3436   \bbl@forkv{#1}{%
3437     \bbl@ifunset{bbl@ADJ@##1@##2}%
3438       {\bbl@cs{ADJ@##1}{##2}}%
3439       {\bbl@cs{ADJ@##1@##2}}}}
3440 %
3441 \def\bbl@adjust@lua#1#2{%
3442   \ifvmode
3443     \ifnum\currentgrouplevel=\z@
3444       \directlua{ Babel.#2 }%
3445       \expandafter\expandafter\expandafter\@gobble
3446     \fi
3447   \fi
3448   {\bbl@error{adjust-only-vertical}{#1}{}{}}}% Gobbled if everything went ok.
3449 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3450   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3451 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3452   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3453 \@namedef{bbl@ADJ@bidi.text@on}{%
3454   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3455 \@namedef{bbl@ADJ@bidi.text@off}{%
3456   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3457 \@namedef{bbl@ADJ@bidi.math@on}{%
3458   \let\bbl@noamsmath\@empty}
3459 \@namedef{bbl@ADJ@bidi.math@off}{%
3460   \let\bbl@noamsmath\relax}
3461 %
3462 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3463   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3464 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3465   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3466 %
3467 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3468   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3469 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3470   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3471 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3472   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3473 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3474   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3475 \@namedef{bbl@ADJ@justify.arabic@on}{%
3476   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
```

```
3477 \@namedef{bbl@ADJ@justify.arabic@off}{%
3478   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3479 %
3480 \def\bbl@adjust@layout#1{%
3481   \ifvmode
3482     #1%
3483     \expandafter\@gobble
3484   \fi
3485   {\bbl@error{layout-only-vertical}{}{}{}}}% Gobbled if everything went ok.
3486 \@namedef{bbl@ADJ@layout.tabular@on}{%
3487   \ifnum\bbl@tabular@mode=\tw@
3488     \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3489   \else
3490     \chardef\bbl@tabular@mode\@ne
3491   \fi}
3492 \@namedef{bbl@ADJ@layout.tabular@off}{%
3493   \ifnum\bbl@tabular@mode=\tw@
3494     \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}%
3495   \else
3496     \chardef\bbl@tabular@mode\z@
3497   \fi}
3498 \@namedef{bbl@ADJ@layout.lists@on}{%
3499   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3500 \@namedef{bbl@ADJ@layout.lists@off}{%
3501   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3502 %
3503 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3504   \bbl@bcpallowedtrue}
3505 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3506   \bbl@bcpallowedfalse}
3507 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3508   \def\bbl@bcp@prefix{#1}}
3509 \def\bbl@bcp@prefix{bcp47-}
3510 \@namedef{bbl@ADJ@autoload.options}#1{%
3511   \def\bbl@autoload@options{#1}}
3512 \let\bbl@autoload@bcpoptions\@empty
3513 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3514   \def\bbl@autoload@bcpoptions{#1}}
3515 \newif\ifbbl@bcptoname
3516 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3517   \bbl@bcptonametrue
3518   \BabelEnsureInfo}
3519 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3520   \bbl@bcptonamefalse}
3521 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3522   \directlua{ Babel.ignore_pre_char = function(node)
3523       return (node.lang == \the\csname l@nohyphenation\endcsname)
3524     end }}
3525 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3526   \directlua{ Babel.ignore_pre_char = function(node)
3527       return false
3528     end }}
3529 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3530   \def\bbl@ignoreinterchar{%
3531     \ifnum\language=\l@nohyphenation
3532       \expandafter\@gobble
3533     \else
3534       \expandafter\@firstofone
3535     \fi}}
3536 \@namedef{bbl@ADJ@interchar.disable@off}{%
3537   \let\bbl@ignoreinterchar\@firstofone}
3538 \@namedef{bbl@ADJ@select.write@shift}{%
3539   \let\bbl@restorelastskip\relax
```

```
3540  \def\bbl@savelastskip{%
3541    \let\bbl@restorelastskip\relax
3542    \ifvmode
3543      \ifdim\lastskip=\z@
3544        \let\bbl@restorelastskip\nobreak
3545      \else
3546        \bbl@exp{%
3547          \def\\\bbl@restorelastskip{%
3548            \skip@=\the\lastskip
3549            \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3550      \fi
3551    \fi}}
3552 \@namedef{bbl@ADJ@select.write@keep}{%
3553    \let\bbl@restorelastskip\relax
3554    \let\bbl@savelastskip\relax}
3555 \@namedef{bbl@ADJ@select.write@omit}{%
3556    \AddBabelHook{babel-select}{beforestart}{%
3557      \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3558    \let\bbl@restorelastskip\relax
3559    \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3560 \@namedef{bbl@ADJ@select.encoding@off}{%
3561    \let\bbl@encoding@select@off\@empty}
```

5.1. Cross referencing macros

The LATEX book states:

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

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

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

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

```
3562 ⟨⟨*More package options⟩⟩ ≡
3563 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3564 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3565 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3566 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3567 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3568 ⟨⟨/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.

```
3569 \bbl@trace{Cross referencing macros}
3570 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3571   \def\@newl@bel#1#2#3{%
3572     {\@safe@activestrue
3573      \bbl@ifunset{#1@#2}%
3574        \relax
3575        {\gdef\@multiplelabels{%
3576           \@latex@warning@no@line{There were multiply-defined labels}}%
3577         \@latex@warning@no@line{Label `#2' multiply defined}}%
3578      \global\@namedef{#1@#2}{#3}}}
```

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

```
3579 \CheckCommand*\@testdef[3]{%
3580   \def\reserved@a{#3}%
3581   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
```

```
3582        \else
3583          \@tempswatrue
3584        \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.

```
3585  \def\@testdef#1#2#3{%   TODO. With @samestring?
3586        \@safe@activestrue
3587        \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3588        \def\bbl@tempb{#3}%
3589        \@safe@activesfalse
3590        \ifx\bbl@tempa\relax
3591        \else
3592          \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3593        \fi
3594        \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3595        \ifx\bbl@tempa\bbl@tempb
3596        \else
3597          \@tempswatrue
3598        \fi}
3599  \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.

```
3600  \bbl@xin@{R}\bbl@opt@safe
3601  \ifin@
3602    \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3603    \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3604      {\expandafter\strip@prefix\meaning\ref}%
3605    \ifin@
3606      \bbl@redefine\@kernel@ref#1{%
3607        \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3608      \bbl@redefine\@kernel@pageref#1{%
3609        \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3610      \bbl@redefine\@kernel@sref#1{%
3611        \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3612      \bbl@redefine\@kernel@spageref#1{%
3613        \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3614    \else
3615      \bbl@redefinerobust\ref#1{%
3616        \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3617      \bbl@redefinerobust\pageref#1{%
3618        \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3619    \fi
3620  \else
3621    \let\org@ref\ref
3622    \let\org@pageref\pageref
3623  \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.

```
3624  \bbl@xin@{B}\bbl@opt@safe
3625  \ifin@
3626    \bbl@redefine\@citex[#1]#2{%
3627      \@safe@activestrue\edef\bbl@tempa{#2}\@safe@activesfalse
```

```
3628        \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.)

```
3629    \AtBeginDocument{%
3630      \@ifpackageloaded{natbib}{%
3631      \def\@citex[#1][#2]#3{%
3632        \@safe@activestrue\edef\bbl@tempa{#3}\@safe@activesfalse
3633        \org@@citex[#1][#2]{\bbl@tempa}}%
3634      }{}}
```

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

```
3635    \AtBeginDocument{%
3636      \@ifpackageloaded{cite}{%
3637        \def\@citex[#1]#2{%
3638          \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3639      }{}}
```

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

```
3640    \bbl@redefine\nocite#1{%
3641      \@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.

```
3642    \bbl@redefine\bibcite{%
3643      \bbl@cite@choice
3644      \bibcite}
```

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

```
3645    \def\bbl@bibcite#1#2{%
3646      \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.

```
3647    \def\bbl@cite@choice{%
3648      \global\let\bibcite\bbl@bibcite
3649      \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3650      \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3651      \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.

```
3652    \AtBeginDocument{\bbl@cite@choice}
```

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

```
3653 \bbl@redefine\@bibitem#1{%
3654   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3655 \else
3656   \let\org@nocite\nocite
3657   \let\org@@citex\@citex
3658   \let\org@bibcite\bibcite
3659   \let\org@@bibitem\@bibitem
3660 \fi
```

5.2. Layout

```
3661 \newcommand\BabelPatchSection[1]{%
3662   \@ifundefined{#1}{}{%
3663     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3664     \@namedef{#1}{%
3665       \@ifstar{\bbl@presec@s{#1}}%
3666                {\@dblarg{\bbl@presec@x{#1}}}}}}
3667 \def\bbl@presec@x#1[#2]#3{%
3668   \bbl@exp{%
3669     \\\select@language@x{\bbl@main@language}%
3670     \\\bbl@cs{sspre@#1}%
3671     \\\bbl@cs{ss@#1}%
3672       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
3673       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
3674     \\\select@language@x{\languagename}}}
3675 \def\bbl@presec@s#1#2{%
3676   \bbl@exp{%
3677     \\\select@language@x{\bbl@main@language}%
3678     \\\bbl@cs{sspre@#1}%
3679     \\\bbl@cs{ss@#1}*%
3680       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
3681     \\\select@language@x{\languagename}}}
3682 \IfBabelLayout{sectioning}%
3683   {\BabelPatchSection{part}%
3684    \BabelPatchSection{chapter}%
3685    \BabelPatchSection{section}%
3686    \BabelPatchSection{subsection}%
3687    \BabelPatchSection{subsubsection}%
3688    \BabelPatchSection{paragraph}%
3689    \BabelPatchSection{subparagraph}%
3690    \def\babel@toc#1{%
3691      \select@language@x{\bbl@main@language}}}{}
3692 \IfBabelLayout{captions}%
3693   {\BabelPatchSection{caption}}{}
```

5.3. Marks

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

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

```
3694 \bbl@trace{Marks}
3695 \IfBabelLayout{sectioning}
3696   {\ifx\bbl@opt@headfoot\@nnil
3697     \g@addto@macro\@resetactivechars{%
3698       \set@typeset@protect
3699       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3700       \let\protect\noexpand
3701       \ifcase\bbl@bidimode\else % Only with bidi. See also above
```

```
3702        \edef\thepage{%
3703          \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3704      \fi}%
3705    \fi}
3706  {\ifbbl@single\else
3707    \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3708    \markright#1{%
3709      \bbl@ifblank{#1}%
3710        {\org@markright{}}%
3711        {\toks@{#1}%
3712          \bbl@exp{%
3713            \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3714              {\\\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, LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```
3715    \ifx\@mkboth\markboth
3716      \def\bbl@tempc{\let\@mkboth\markboth}%
3717    \else
3718      \def\bbl@tempc{}%
3719    \fi
3720    \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3721    \markboth#1#2{%
3722      \protected@edef\bbl@tempb##1{%
3723        \protect\foreignlanguage
3724        {\languagename}{\protect\bbl@restore@actives##1}}%
3725      \bbl@ifblank{#1}%
3726        {\toks@{}}%
3727        {\toks@\expandafter{\bbl@tempb{#1}}}%
3728      \bbl@ifblank{#2}%
3729        {\@temptokena{}}%
3730        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3731      \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3732      \bbl@tempc
3733    \fi}  % end ifbbl@single, end \IfBabelLayout
```

5.4.   Other packages

5.4.1.   `ifthen`

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

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

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

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

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

```
3734 \bbl@trace{Preventing clashes with other packages}
```

```
3735 \ifx\org@ref\@undefined\else
3736   \bbl@xin@{R}\bbl@opt@safe
3737   \ifin@
3738     \AtBeginDocument{%
3739       \@ifpackageloaded{ifthen}{%
3740         \bbl@redefine@long\ifthenelse#1#2#3{%
3741           \let\bbl@temp@pref\pageref
3742           \let\pageref\org@pageref
3743           \let\bbl@temp@ref\ref
3744           \let\ref\org@ref
3745           \@safe@activestrue
3746           \org@ifthenelse{#1}%
3747             {\let\pageref\bbl@temp@pref
3748              \let\ref\bbl@temp@ref
3749              \@safe@activesfalse
3750              #2}%
3751             {\let\pageref\bbl@temp@pref
3752              \let\ref\bbl@temp@ref
3753              \@safe@activesfalse
3754              #3}%
3755         }%
3756       }{}%
3757     }
3758 \fi
```

5.4.2. `varioref`

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

```
3759   \AtBeginDocument{%
3760     \@ifpackageloaded{varioref}{%
3761       \bbl@redefine\@@vpageref#1[#2]#3{%
3762         \@safe@activestrue
3763         \org@@@vpageref{#1}[#2]{#3}%
3764         \@safe@activesfalse}%
3765       \bbl@redefine\vrefpagenum#1#2{%
3766         \@safe@activestrue
3767         \org@vrefpagenum{#1}{#2}%
3768         \@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.

```
3769     \expandafter\def\csname Ref \endcsname#1{%
3770       \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3771     }{}%
3772   }
3773 \fi
```

5.4.3. `hhline`

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

```
3774 \AtEndOfPackage{%
```

```
3775    \AtBeginDocument{%
3776      \@ifpackageloaded{hhline}%
3777        {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3778         \else
3779           \makeatletter
3780           \def\@currname{hhline}\input{hhline.sty}\makeatother
3781        \fi}%
3782      {}}}
```

\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 LaTeX (\DeclareFontFamilySubstitution).

```
3783 \def\substitutefontfamily#1#2#3{%
3784   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3785   \immediate\write15{%
3786     \string\ProvidesFile{#1#2.fd}%
3787     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3788      \space generated font description file]^^J
3789     \string\DeclareFontFamily{#1}{#2}{}^^J
3790     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3791     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3792     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3793     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3794     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3795     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3796     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3797     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3798     }%
3799   \closeout15
3800   }
3801 \@onlypreamble\substitutefontfamily
```

5.5.   Encoding and fonts

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

\ensureascii
```
3802 \bbl@trace{Encoding and fonts}
3803 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3804 \newcommand\BabelNonText{TS1,T3,TS3}
3805 \let\org@TeX\TeX
3806 \let\org@LaTeX\LaTeX
3807 \let\ensureascii\@firstofone
3808 \let\asciiencoding\@empty
3809 \AtBeginDocument{%
3810   \def\@elt#1{,#1,}%
3811   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3812   \let\@elt\relax
3813   \let\bbl@tempb\@empty
3814   \def\bbl@tempc{OT1}%
3815   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3816     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3817   \bbl@foreach\bbl@tempa{%
3818     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3819     \ifin@
3820       \def\bbl@tempb{#1}% Store last non-ascii
3821     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3822       \ifin@\else
```

```
3823        \def\bbl@tempc{#1}% Store last ascii
3824      \fi
3825    \fi}%
3826  \ifx\bbl@tempb\@empty\else
3827    \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3828    \ifin@\else
3829      \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3830    \fi
3831    \let\asciiencoding\bbl@tempc
3832    \renewcommand\ensureascii[1]{%
3833      {\fontencoding{\asciiencoding}\selectfont#1}}%
3834    \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3835    \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3836  \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.

```
3837 \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.

```
3838 \AtBeginDocument{%
3839  \@ifpackageloaded{fontspec}%
3840    {\xdef\latinencoding{%
3841       \ifx\UTFencname\@undefined
3842         EU\ifcase\bbl@engine\or2\or1\fi
3843       \else
3844         \UTFencname
3845       \fi}}%
3846    {\gdef\latinencoding{OT1}%
3847     \ifx\cf@encoding\bbl@t@one
3848       \xdef\latinencoding{\bbl@t@one}%
3849     \else
3850       \def\@elt#1{,#1,}%
3851       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3852       \let\@elt\relax
3853       \bbl@xin@{,T1,}\bbl@tempa
3854       \ifin@
3855         \xdef\latinencoding{\bbl@t@one}%
3856       \fi
3857     \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.

```
3858 \DeclareRobustCommand{\latintext}{%
3859  \fontencoding{\latinencoding}\selectfont
3860  \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.

```
3861 \ifx\@undefined\DeclareTextFontCommand
3862  \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3863 \else
3864  \DeclareTextFontCommand{\textlatin}{\latintext}
3865 \fi
```

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

```
3866 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}
```

5.6. Basic bidi support

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

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

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

- pdftex provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.

- xetex is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour TeX grouping.

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

```
3867 \bbl@trace{Loading basic (internal) bidi support}
3868 \ifodd\bbl@engine
3869 \else % TODO. Move to txtbabel. Any xe+lua bidi
3870   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3871     \bbl@error{bidi-only-lua}{}{}{}%
3872     \let\bbl@beforeforeign\leavevmode
3873     \AtEndOfPackage{%
3874       \EnableBabelHook{babel-bidi}%
3875       \bbl@xebidipar}
3876   \fi\fi
3877   \def\bbl@loadxebidi#1{%
3878     \ifx\RTLfootnotetext\@undefined
3879       \AtEndOfPackage{%
3880         \EnableBabelHook{babel-bidi}%
3881         \ifx\fontspec\@undefined
3882           \usepackage{fontspec}% bidi needs fontspec
3883         \fi
3884         \usepackage#1{bidi}%
3885         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3886         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3887           \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
3888             \bbl@digitsdotdash % So ignore in 'R' bidi
3889         \fi}}%
3890     \fi}
3891   \ifnum\bbl@bidimode>200 % Any xe bidi=
3892     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3893       \bbl@tentative{bidi=bidi}
3894       \bbl@loadxebidi{}
3895     \or
3896       \bbl@loadxebidi{[rldocument]}
3897     \or
3898       \bbl@loadxebidi{}
3899     \fi
3900   \fi
3901 \fi
3902 % TODO? Separate:
```

```
3903 \ifnum\bbl@bidimode=\@ne % bidi=default
3904   \let\bbl@beforeforeign\leavevmode
3905   \ifodd\bbl@engine % lua
3906     \newattribute\bbl@attr@dir
3907     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3908     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3909   \fi
3910   \AtEndOfPackage{%
3911     \EnableBabelHook{babel-bidi}% pdf/lua/xe
3912     \ifodd\bbl@engine\else % pdf/xe
3913       \bbl@xebidipar
3914     \fi}
3915 \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.

```
3916 \bbl@trace{Macros to switch the text direction}
3917 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3918 \def\bbl@rscripts{%
3919   ,Garay,Todhri,Imperial Aramaic,Avestan,Cypriot,Elymaic,Hatran,Hebrew,%
3920   Old Hungarian,Kharoshthi,Lydian,Mandaean,Manichaean,Mende Kikakui,%
3921   Meroitic Cursive,Meroitic,Old North Arabian,Nabataean,N'Ko,%
3922   Old Turkic,Orkhon,Palmyrene,Inscriptional Pahlavi,Psalter Pahlavi,%
3923   Phoenician,Inscriptional Parthian,Hanifi,Samaritan,Old Sogdian,%
3924   Old South Arabian,Yezidi,}%
3925 \def\bbl@provide@dirs#1{%
3926   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3927   \ifin@
3928     \global\bbl@csarg\chardef{wdir@#1}\@ne
3929     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3930     \ifin@
3931       \global\bbl@csarg\chardef{wdir@#1}\tw@
3932     \fi
3933   \else
3934     \global\bbl@csarg\chardef{wdir@#1}\z@
3935   \fi
3936   \ifodd\bbl@engine
3937     \bbl@csarg\ifcase{wdir@#1}%
3938       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3939     \or
3940       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3941     \or
3942       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3943     \fi
3944   \fi}
3945 \def\bbl@switchdir{%
3946   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3947   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3948   \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
3949 \def\bbl@setdirs#1{% TODO - math
3950   \ifcase\bbl@select@type % TODO - strictly, not the right test
3951     \bbl@bodydir{#1}%
3952     \bbl@pardir{#1}% <- Must precede \bbl@textdir
3953   \fi
3954   \bbl@textdir{#1}}
3955 \ifnum\bbl@bidimode>\z@
3956   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3957   \DisableBabelHook{babel-bidi}
3958 \fi
```

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

```
3959 \ifodd\bbl@engine  % luatex=1
3960 \else % pdftex=0, xetex=2
```

```
3961  \newcount\bbl@dirlevel
3962  \chardef\bbl@thetextdir\z@
3963  \chardef\bbl@thepardir\z@
3964  \def\bbl@textdir#1{%
3965    \ifcase#1\relax
3966      \chardef\bbl@thetextdir\z@
3967      \@nameuse{setlatin}%
3968      \bbl@textdir@i\beginL\endL
3969    \else
3970      \chardef\bbl@thetextdir\@ne
3971      \@nameuse{setnonlatin}%
3972      \bbl@textdir@i\beginR\endR
3973    \fi}
3974  \def\bbl@textdir@i#1#2{%
3975    \ifhmode
3976      \ifnum\currentgrouplevel>\z@
3977        \ifnum\currentgrouplevel=\bbl@dirlevel
3978          \bbl@error{multiple-bidi}{}{}{}%
3979          \bgroup\aftergroup#2\aftergroup\egroup
3980        \else
3981          \ifcase\currentgrouptype\or % 0 bottom
3982            \aftergroup#2% 1 simple {}
3983          \or
3984            \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3985          \or
3986            \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
3987          \or\or\or % vbox vtop align
3988          \or
3989            \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3990          \or\or\or\or\or\or % output math disc insert vcent mathchoice
3991          \or
3992            \aftergroup#2% 14 \begingroup
3993          \else
3994            \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3995          \fi
3996        \fi
3997        \bbl@dirlevel\currentgrouplevel
3998      \fi
3999      #1%
4000    \fi}
4001  \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4002  \let\bbl@bodydir\@gobble
4003  \let\bbl@pagedir\@gobble
4004  \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).

```
4005  \def\bbl@xebidipar{%
4006    \let\bbl@xebidipar\relax
4007    \TeXXeTstate\@ne
4008    \def\bbl@xeeverypar{%
4009      \ifcase\bbl@thepardir
4010        \ifcase\bbl@thetextdir\else\beginR\fi
4011      \else
4012        {\setbox\z@\lastbox\beginR\box\z@}%
4013      \fi}%
4014    \AddToHook{para/begin}{\bbl@xeeverypar}}
4015  \ifnum\bbl@bidimode>200 % Any xe bidi=
4016    \let\bbl@textdir@i\@gobbletwo
4017    \let\bbl@xebidipar\@empty
4018    \AddBabelHook{bidi}{foreign}{%
4019      \ifcase\bbl@thetextdir
```

```
4020        \BabelWrapText{\LR{##1}}%
4021      \else
4022        \BabelWrapText{\RL{##1}}%
4023      \fi}
4024    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4025  \fi
4026 \fi
```

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

```
4027 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4028 \AtBeginDocument{%
4029  \ifx\pdfstringdefDisableCommands\@undefined\else
4030    \ifx\pdfstringdefDisableCommands\relax\else
4031      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4032    \fi
4033  \fi}
```

5.7. Local Language Configuration

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

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

```
4034 \bbl@trace{Local Language Configuration}
4035 \ifx\loadlocalcfg\@undefined
4036  \@ifpackagewith{babel}{noconfigs}%
4037    {\let\loadlocalcfg\@gobble}%
4038    {\def\loadlocalcfg#1{%
4039      \InputIfFileExists{#1.cfg}%
4040        {\typeout{**************************************^^J%
4041                  * Local config file #1.cfg used^^J%
4042                  *}}%
4043      \@empty}}
4044 \fi
```

5.8. Language options

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

```
4045 \bbl@trace{Language options}
4046 \let\bbl@afterlang\relax
4047 \let\BabelModifiers\relax
4048 \let\bbl@loaded\@empty
4049 \def\bbl@load@language#1{%
4050  \InputIfFileExists{#1.ldf}%
4051    {\edef\bbl@loaded{\CurrentOption
4052      \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4053    \expandafter\let\expandafter\bbl@afterlang
4054      \csname\CurrentOption.ldf-h@@k\endcsname
4055    \expandafter\let\expandafter\BabelModifiers
4056      \csname bbl@mod@\CurrentOption\endcsname
4057    \bbl@exp{\\\AtBeginDocument{%
4058      \\\bbl@usehooks@lang{\CurrentOption}{begindocument}{{\CurrentOption}}}}}%
4059    {\IfFileExists{babel-#1.tex}%
4060      {\def\bbl@tempa{%
4061        .\\There is a locale ini file for this language.\\%
4062        If it's the main language, try adding `provide=*'\\%
4063        to the babel package options}}%
4064      {\let\bbl@tempa\empty}%
4065    \bbl@error{unknown-package-option}{}{}{}}}
```

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

```
4066 \def\bbl@try@load@lang#1#2#3{%
4067   \IfFileExists{\CurrentOption.ldf}%
4068     {\bbl@load@language{\CurrentOption}}%
4069     {#1\bbl@load@language{#2}#3}}
4070 %
4071 \DeclareOption{friulian}{\bbl@try@load@lang{}{friulan}{}}
4072 \DeclareOption{hebrew}{%
4073   \ifcase\bbl@engine\or
4074     \bbl@error{only-pdftex-lang}{hebrew}{luatex}{}%
4075   \fi
4076   \input{rlbabel.def}%
4077   \bbl@load@language{hebrew}}
4078 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4079 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4080 % \DeclareOption{northernkurdish}{\bbl@try@load@lang{}{kurmanji}{}}
4081 \DeclareOption{polutonikogreek}{%
4082   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4083 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4084 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4085 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}
```

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

```
4086 \ifx\bbl@opt@config\@nnil
4087   \@ifpackagewith{babel}{noconfigs}{}%
4088     {\InputIfFileExists{bblopts.cfg}%
4089       {\typeout{*************************************^^J%
4090                * Local config file bblopts.cfg used^^J%
4091                *}}%
4092       {}}%
4093 \else
4094   \InputIfFileExists{\bbl@opt@config.cfg}%
4095     {\typeout{*************************************^^J%
4096              * Local config file \bbl@opt@config.cfg used^^J%
4097              *}}%
4098     {\bbl@error{config-not-found}{}{}{}}%
4099 \fi
```

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

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

```
4100 \def\bbl@tempf{,}
4101 \bbl@foreach\@raw@classoptionslist{%
4102   \in@{=}{#1}%
4103   \ifin@\else
4104     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4105   \fi}
4106 \ifx\bbl@opt@main\@nnil
4107   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4108     \let\bbl@tempb\@empty
4109     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4110     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
```

```
4111  \bbl@foreach\bbl@tempb{%    \bbl@tempb is a reversed list
4112    \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4113      \ifodd\bbl@iniflag % = *=
4114        \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4115      \else % n +=
4116        \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4117      \fi
4118    \fi}%
4119  \fi
4120 \else
4121  \bbl@info{Main language set with 'main='. Except if you have\\%
4122            problems, prefer the default mechanism for setting\\%
4123            the main language, ie, as the last declared.\\%
4124            Reported}
4125 \fi
```

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

```
4126 \ifx\bbl@opt@main\@nnil\else
4127  \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4128  \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4129 \fi
```

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

```
4130 \bbl@foreach\bbl@language@opts{%
4131  \def\bbl@tempa{#1}%
4132  \ifx\bbl@tempa\bbl@opt@main\else
4133    \ifnum\bbl@iniflag<\tw@     % 0 ø (other = ldf)
4134      \bbl@ifunset{ds@#1}%
4135        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4136        {}%
4137    \else                      % + * (other = ini)
4138      \DeclareOption{#1}{%
4139        \bbl@ldfinit
4140        \babelprovide[@import]{#1}% %%%%
4141        \bbl@afterldf{}}%
4142    \fi
4143  \fi}
4144 \bbl@foreach\bbl@tempf{%
4145  \def\bbl@tempa{#1}%
4146  \ifx\bbl@tempa\bbl@opt@main\else
4147    \ifnum\bbl@iniflag<\tw@     % 0 ø (other = ldf)
4148      \bbl@ifunset{ds@#1}%
4149        {\IfFileExists{#1.ldf}%
4150          {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4151          {}}%
4152        {}%
4153    \else                      % + * (other = ini)
4154      \IfFileExists{babel-#1.tex}%
4155        {\DeclareOption{#1}{%
4156          \bbl@ldfinit
4157          \babelprovide[@import]{#1}%  %%%%%
4158          \bbl@afterldf{}}}%
4159        {}%
4160    \fi
4161  \fi}
```

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

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

```
4162 \NewHook{babel/presets}
```

```
4163 \UseHook{babel/presets}
4164 \def\AfterBabelLanguage#1{%
4165   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4166 \DeclareOption*{}
4167 \ProcessOptions*
```

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

```
4168 \bbl@trace{Option 'main'}
4169 \ifx\bbl@opt@main\@nnil
4170   \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4171   \let\bbl@tempc\@empty
4172   \edef\bbl@templ{,\bbl@loaded,}
4173   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4174   \bbl@for\bbl@tempb\bbl@tempa{%
4175     \edef\bbl@tempd{,\bbl@tempb,}%
4176     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4177     \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4178     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4179   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4180   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4181   \ifx\bbl@tempb\bbl@tempc\else
4182     \bbl@warning{%
4183       Last declared language option is '\bbl@tempc',\\%
4184       but the last processed one was '\bbl@tempb'.\\%
4185       The main language can't be set as both a global\\%
4186       and a package option. Use 'main=\bbl@tempc' as\\%
4187       option. Reported}
4188   \fi
4189 \else
4190   \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4191     \bbl@ldfinit
4192     \let\CurrentOption\bbl@opt@main
4193     \bbl@exp{%  \bbl@opt@provide = empty if *
4194       \\\babelprovide
4195         [\bbl@opt@provide,@import,main]%  %%%%
4196         {\bbl@opt@main}}%
4197     \bbl@afterldf{}
4198     \DeclareOption{\bbl@opt@main}{}
4199   \else % case 0,2 (main is ldf)
4200     \ifx\bbl@loadmain\relax
4201       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4202     \else
4203       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4204     \fi
4205     \ExecuteOptions{\bbl@opt@main}
4206     \@namedef{ds@\bbl@opt@main}{}%
4207   \fi
4208   \DeclareOption*{}
4209   \ProcessOptions*
4210 \fi
4211 \bbl@exp{%
4212   \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{begindocument}{{}}}}%
4213 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}{}{}}
```

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

```
4214 \ifx\bbl@main@language\@undefined
4215   \bbl@info{%
4216     You haven't specified a language as a class or package\\%
```

```
4217    option. I'll load 'nil'. Reported}
4218    \bbl@load@language{nil}
4219 \fi
4220 ⟨/package⟩
```

6.   The kernel of Babel

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

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

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

A proxy file for switch.def

```
4221 ⟨*kernel⟩
4222 \let\bbl@onlyswitch\@empty
4223 \input babel.def
4224 \let\bbl@onlyswitch\@undefined
4225 ⟨/kernel⟩
```

7.   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.

```
4226 ⟨*errors⟩
4227 \catcode`\{=1  \catcode`\}=2  \catcode`\#=6
4228 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4229 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4230 \catcode`\@=11 \catcode`\^=7
4231 %
4232 \ifx\MessageBreak\@undefined
4233   \gdef\bbl@error@i#1#2{%
4234     \begingroup
4235       \newlinechar=`\^^J
4236       \def\\{^^J(babel) }%
4237       \errhelp{#2}\errmessage{\\#1}%
4238     \endgroup}
4239 \else
4240   \gdef\bbl@error@i#1#2{%
4241     \begingroup
4242       \def\\{\MessageBreak}%
4243       \PackageError{babel}{#1}{#2}%
4244     \endgroup}
4245 \fi
4246 \def\bbl@errmessage#1#2#3{%
4247   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4248     \bbl@error@i{#2}{#3}}}
4249 % Implicit #2#3#4:
4250 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4251 %
4252 \bbl@errmessage{not-yet-available}
4253     {Not yet available}%
4254     {Find an armchair, sit down and wait}
4255 \bbl@errmessage{bad-package-option}%
4256     {Bad option '#1=#2'. Either you have misspelled the\\%
```

```
4257    key or there is a previous setting of '#1'. Valid\\%
4258    keys are, among others, 'shorthands', 'main', 'bidi',\\%
4259    'strings', 'config', 'headfoot', 'safe', 'math'.}%
4260   {See the manual for further details.}
4261 \bbl@errmessage{base-on-the-fly}
4262   {For a language to be defined on the fly 'base'\\%
4263    is not enough, and the whole package must be\\%
4264    loaded. Either delete the 'base' option or\\%
4265    request the languages explicitly}%
4266   {See the manual for further details.}
4267 \bbl@errmessage{undefined-language}
4268   {You haven't defined the language '#1' yet.\\%
4269    Perhaps you misspelled it or your installation\\%
4270    is not complete}%
4271   {Your command will be ignored, type <return> to proceed}
4272 \bbl@errmessage{shorthand-is-off}
4273   {I can't declare a shorthand turned off (\string#2)}
4274   {Sorry, but you can't use shorthands which have been\\%
4275    turned off in the package options}
4276 \bbl@errmessage{not-a-shorthand}
4277   {The character '\string #1' should be made a shorthand character;\\%
4278    add the command \string\useshorthands\string{#1\string} to
4279    the preamble.\\%
4280    I will ignore your instruction}%
4281   {You may proceed, but expect unexpected results}
4282 \bbl@errmessage{not-a-shorthand-b}
4283   {I can't switch '\string#2' on or off--not a shorthand}%
4284   {This character is not a shorthand. Maybe you made\\%
4285    a typing mistake? I will ignore your instruction.}
4286 \bbl@errmessage{unknown-attribute}
4287   {The attribute #2 is unknown for language #1.}%
4288   {Your command will be ignored, type <return> to proceed}
4289 \bbl@errmessage{missing-group}
4290   {Missing group for string \string#1}%
4291   {You must assign strings to some category, typically\\%
4292    captions or extras, but you set none}
4293 \bbl@errmessage{only-lua-xe}
4294   {This macro is available only in LuaLaTeX and XeLaTeX.}%
4295   {Consider switching to these engines.}
4296 \bbl@errmessage{only-lua}
4297   {This macro is available only in LuaLaTeX}%
4298   {Consider switching to that engine.}
4299 \bbl@errmessage{unknown-provide-key}
4300   {Unknown key '#1' in \string\babelprovide}%
4301   {See the manual for valid keys}%
4302 \bbl@errmessage{unknown-mapfont}
4303   {Option '\bbl@KVP@mapfont' unknown for\\%
4304    mapfont. Use 'direction'}%
4305   {See the manual for details.}
4306 \bbl@errmessage{no-ini-file}
4307   {There is no ini file for the requested language\\%
4308    (#1: \languagename). Perhaps you misspelled it or your\\%
4309    installation is not complete}%
4310   {Fix the name or reinstall babel.}
4311 \bbl@errmessage{digits-is-reserved}
4312   {The counter name 'digits' is reserved for mapping\\%
4313    decimal digits}%
4314   {Use another name.}
4315 \bbl@errmessage{limit-two-digits}
4316   {Currently two-digit years are restricted to the\\
4317    range 0-9999}%
4318   {There is little you can do. Sorry.}
4319 \bbl@errmessage{alphabetic-too-large}
```

```
4320 {Alphabetic numeral too large (#1)}%
4321 {Currently this is the limit.}
4322 \bbl@errmessage{no-ini-info}
4323   {I've found no info for the current locale.\\%
4324    The corresponding ini file has not been loaded\\%
4325    Perhaps it doesn't exist}%
4326   {See the manual for details.}
4327 \bbl@errmessage{unknown-ini-field}
4328   {Unknown field '#1' in \string\BCPdata.\\%
4329    Perhaps you misspelled it}%
4330   {See the manual for details.}
4331 \bbl@errmessage{unknown-locale-key}
4332   {Unknown key for locale '#2':\\%
4333    #3\\%
4334    \string#1 will be set to \string\relax}%
4335   {Perhaps you misspelled it.}%
4336 \bbl@errmessage{adjust-only-vertical}
4337   {Currently, #1 related features can be adjusted only\\%
4338    in the main vertical list}%
4339   {Maybe things change in the future, but this is what it is.}
4340 \bbl@errmessage{layout-only-vertical}
4341   {Currently, layout related features can be adjusted only\\%
4342    in vertical mode}%
4343   {Maybe things change in the future, but this is what it is.}
4344 \bbl@errmessage{bidi-only-lua}
4345   {The bidi method 'basic' is available only in\\%
4346    luatex. I'll continue with 'bidi=default', so\\%
4347    expect wrong results}%
4348   {See the manual for further details.}
4349 \bbl@errmessage{multiple-bidi}
4350   {Multiple bidi settings inside a group}%
4351   {I'll insert a new group, but expect wrong results.}
4352 \bbl@errmessage{unknown-package-option}
4353   {Unknown option '\CurrentOption'. Either you misspelled it\\%
4354    or the language definition file \CurrentOption.ldf\\%
4355    was not found%
4356    \bbl@tempa}
4357   {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4358    activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4359    headfoot=, strings=, config=, hyphenmap=, or a language name.}
4360 \bbl@errmessage{config-not-found}
4361   {Local config file '\bbl@opt@config.cfg' not found}%
4362   {Perhaps you misspelled it.}
4363 \bbl@errmessage{late-after-babel}
4364   {Too late for \string\AfterBabelLanguage}%
4365   {Languages have been loaded, so I can do nothing}
4366 \bbl@errmessage{double-hyphens-class}
4367   {Double hyphens aren't allowed in \string\babelcharclass\\%
4368    because it's potentially ambiguous}%
4369   {See the manual for further info}
4370 \bbl@errmessage{unknown-interchar}
4371   {'#1' for '\languagename' cannot be enabled.\\%
4372    Maybe there is a typo}%
4373   {See the manual for further details.}
4374 \bbl@errmessage{unknown-interchar-b}
4375   {'#1' for '\languagename' cannot be disabled.\\%
4376    Maybe there is a typo}%
4377   {See the manual for further details.}
4378 \bbl@errmessage{charproperty-only-vertical}
4379   {\string\babelcharproperty\space can be used only in\\%
4380    vertical mode (preamble or between paragraphs)}%
4381   {See the manual for further info}
4382 \bbl@errmessage{unknown-char-property}
```

```
4383     {No property named '#2'. Allowed values are\\%
4384      direction (bc), mirror (bmg), and linebreak (lb)}%
4385     {See the manual for further info}
4386 \bbl@errmessage{bad-transform-option}
4387     {Bad option '#1' in a transform.\\%
4388      I'll ignore it but expect more errors}%
4389     {See the manual for further info.}
4390 \bbl@errmessage{font-conflict-transforms}
4391     {Transforms cannot be re-assigned to different\\%
4392      fonts. The conflict is in '\bbl@kv@label'.\\%
4393      Apply the same fonts or use a different label}%
4394     {See the manual for further details.}
4395 \bbl@errmessage{transform-not-available}
4396     {'#1' for '\languagename' cannot be enabled.\\%
4397      Maybe there is a typo or it's a font-dependent transform}%
4398     {See the manual for further details.}
4399 \bbl@errmessage{transform-not-available-b}
4400     {'#1' for '\languagename' cannot be disabled.\\%
4401      Maybe there is a typo or it's a font-dependent transform}%
4402     {See the manual for further details.}
4403 \bbl@errmessage{year-out-range}
4404     {Year out of range.\\%
4405      The allowed range is #1}%
4406     {See the manual for further details.}
4407 \bbl@errmessage{only-pdftex-lang}
4408     {The '#1' ldf style doesn't work with #2,\\%
4409      but you can use the ini locale instead.\\%
4410      Try adding 'provide=*' to the option list. You may\\%
4411      also want to set 'bidi=' to some value}%
4412     {See the manual for further details.}
4413 \bbl@errmessage{hyphenmins-args}
4414     {\string\babelhyphenmins\ accepts either the optional\\%
4415      argument or the star, but not both at the same time}%
4416     {See the manual for further details.}
4417 ⟨/errors⟩
4418 ⟨∗patterns⟩
```

8. Loading hyphenation patterns

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

```
4419 <@Make sure ProvidesFile is defined@>
4420 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4421 \xdef\bbl@format{\jobname}
4422 \def\bbl@version{<@version@>}
4423 \def\bbl@date{<@date@>}
4424 \ifx\AtBeginDocument\@undefined
4425   \def\@empty{}
4426 \fi
4427 <@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.

```
4428 \def\process@line#1#2 #3 #4 {%
4429   \ifx=#1%
4430     \process@synonym{#2}%
4431   \else
4432     \process@language{#1#2}{#3}{#4}%
4433   \fi
```

```
4434   \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.

```
4435 \toks@{}
4436 \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.

```
4437 \def\process@synonym#1{%
4438   \ifnum\last@language=\m@ne
4439     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4440   \else
4441     \expandafter\chardef\csname l@#1\endcsname\last@language
4442     \wlog{\string\l@#1=\string\language\the\last@language}%
4443     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4444       \csname\languagename hyphenmins\endcsname
4445     \let\bbl@elt\relax
4446     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4447   \fi}
```

\process@language   The macro \process@language is used to process a non-empty line from the
'configuration file'. It has three arguments, each delimited by white space. The first argument is the
'name' of a language; the second is the name of the file that contains the patterns. The optional third
argument is the name of a file containing hyphenation exceptions.
    The first thing to do is call \addlanguage to allocate a pattern register and to make that register
'active'. Then the pattern file is read.
    For some hyphenation patterns it is needed to load them with a specific font encoding selected.
This can be specified in the file language.dat by adding for instance ':T1' to the name of the
language. The macro \bbl@get@enc extracts the font encoding from the language name and stores it
in \bbl@hyph@enc. The latter can be used in hyphenation files if you need to set a behavior
depending on the given encoding (it is set to empty if no encoding is given).
    Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. TEX does not keep
track of these assignments. Therefore we try to detect such assignments and store them in the
\⟨language⟩hyphenmins macro. When no assignments were made we provide a default setting.
    Some pattern files contain changes to the \lccode en \uccode arrays. Such changes should remain
local to the language; therefore we process the pattern file in a group; the \patterns command acts
globally so its effect will be remembered.
    Then we globally store the settings of \lefthyphenmin and \righthyphenmin and close the group.
    When the hyphenation patterns have been processed we need to see if a file with hyphenation
exceptions needs to be read. This is the case when the third argument is not empty and when it does
not contain a space token. (Note however there is no need to save hyphenation exceptions into the
format.)
    \bbl@languages saves a snapshot of the loaded languages in the form
\bbl@elt{⟨language-name⟩}{⟨number⟩} {⟨patterns-file⟩}{⟨exceptions-file⟩}. Note the last 2
arguments are empty in 'dialects' defined in language.dat with =. Note also the language name can
have encoding info.
    Finally, if the counter \language is equal to zero we execute the synonyms stored.

```
4448 \def\process@language#1#2#3{%
4449   \expandafter\addlanguage\csname l@#1\endcsname
4450   \expandafter\language\csname l@#1\endcsname
4451   \edef\languagename{#1}%
4452   \bbl@hook@everylanguage{#1}%
4453   % > luatex
4454   \bbl@get@enc#1::\@@@
4455   \begingroup
4456     \lefthyphenmin\m@ne
4457     \bbl@hook@loadpatterns{#2}%
4458     % > luatex
```

```
4459    \ifnum\lefthyphenmin=\m@ne
4460    \else
4461      \expandafter\xdef\csname #1hyphenmins\endcsname{%
4462        \the\lefthyphenmin\the\righthyphenmin}%
4463    \fi
4464  \endgroup
4465  \def\bbl@tempa{#3}%
4466  \ifx\bbl@tempa\@empty\else
4467    \bbl@hook@loadexceptions{#3}%
4468    %  > luatex
4469  \fi
4470  \let\bbl@elt\relax
4471  \edef\bbl@languages{%
4472    \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4473  \ifnum\the\language=\z@
4474    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4475      \set@hyphenmins\tw@\thr@@\relax
4476    \else
4477      \expandafter\expandafter\expandafter\set@hyphenmins
4478        \csname #1hyphenmins\endcsname
4479    \fi
4480    \the\toks@
4481    \toks@{}%
4482  \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.

```
4483 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
```

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

```
4484 \def\bbl@hook@everylanguage#1{}
4485 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4486 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4487 \def\bbl@hook@loadkernel#1{%
4488   \def\addlanguage{\csname newlanguage\endcsname}%
4489   \def\adddialect##1##2{%
4490     \global\chardef##1##2\relax
4491     \wlog{\string##1 = a dialect from \string\language##2}}%
4492   \def\iflanguage##1{%
4493     \expandafter\ifx\csname l@##1\endcsname\relax
4494       \@nolanerr{##1}%
4495     \else
4496       \ifnum\csname l@##1\endcsname=\language
4497         \expandafter\expandafter\expandafter\@firstoftwo
4498       \else
4499         \expandafter\expandafter\expandafter\@secondoftwo
4500       \fi
4501     \fi}%
4502   \def\providehyphenmins##1##2{%
4503     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4504       \@namedef{##1hyphenmins}{##2}%
4505     \fi}%
4506   \def\set@hyphenmins##1##2{%
4507     \lefthyphenmin##1\relax
4508     \righthyphenmin##2\relax}%
4509   \def\selectlanguage{%
4510     \errhelp{Selecting a language requires a package supporting it}%
4511     \errmessage{Not loaded}}%
4512   \let\foreignlanguage\selectlanguage
4513   \let\otherlanguage\selectlanguage
```

```
4514 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4515 \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4516 \def\setlocale{%
4517    \errhelp{Find an armchair, sit down and wait}%
4518    \errmessage{(babel) Not yet available}}%
4519 \let\uselocale\setlocale
4520 \let\locale\setlocale
4521 \let\selectlocale\setlocale
4522 \let\localename\setlocale
4523 \let\textlocale\setlocale
4524 \let\textlanguage\setlocale
4525 \let\languagetext\setlocale}
4526 \begingroup
4527 \def\AddBabelHook#1#2{%
4528    \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4529       \def\next{\toks1}%
4530    \else
4531       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4532    \fi
4533    \next}
4534 \ifx\directlua\@undefined
4535    \ifx\XeTeXinputencoding\@undefined\else
4536       \input xebabel.def
4537    \fi
4538 \else
4539    \input luababel.def
4540 \fi
4541 \openin1 = babel-\bbl@format.cfg
4542 \ifeof1
4543 \else
4544    \input babel-\bbl@format.cfg\relax
4545 \fi
4546 \closein1
4547 \endgroup
4548 \bbl@hook@loadkernel{switch.def}
```

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

```
4549 \openin1 = language.dat
```

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

```
4550 \def\languagename{english}%
4551 \ifeof1
4552    \message{I couldn't find the file language.dat,\space
4553            I will try the file hyphen.tex}
4554    \input hyphen.tex\relax
4555    \chardef\l@english\z@
4556 \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.

```
4557    \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.

```
4558    \loop
4559       \endlinechar\m@ne
4560       \read1 to \bbl@line
4561       \endlinechar`\^^M
```

99

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.

```
4562    \if T\ifeof1F\fi T\relax
4563      \ifx\bbl@line\@empty\else
4564        \edef\bbl@line{\bbl@line\space\space\space}%
4565        \expandafter\process@line\bbl@line\relax
4566      \fi
4567    \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.

```
4568    \begingroup
4569      \def\bbl@elt#1#2#3#4{%
4570        \global\language=#2\relax
4571        \gdef\languagename{#1}%
4572        \def\bbl@elt##1##2##3##4{}}%
4573      \bbl@languages
4574    \endgroup
4575 \fi
4576 \closein1
```

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

```
4577 \if/\the\toks@/\else
4578    \errhelp{language.dat loads no language, only synonyms}
4579    \errmessage{Orphan language synonym}
4580 \fi
```

Also remove some macros from memory and raise an error if \toks@ is not empty. Finally load switch.def, but the latter is not required and the line inputting it may be commented out.

```
4581 \let\bbl@line\@undefined
4582 \let\process@line\@undefined
4583 \let\process@synonym\@undefined
4584 \let\process@language\@undefined
4585 \let\bbl@get@enc\@undefined
4586 \let\bbl@hyph@enc\@undefined
4587 \let\bbl@tempa\@undefined
4588 \let\bbl@hook@loadkernel\@undefined
4589 \let\bbl@hook@everylanguage\@undefined
4590 \let\bbl@hook@loadpatterns\@undefined
4591 \let\bbl@hook@loadexceptions\@undefined
4592 ⟨/patterns⟩
```

Here the code for iniTeX ends.

9.   **luatex** + **xetex: common stuff**

Add the bidi handler just before luaoftload, 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).

```
4593 ⟨⟨*More package options⟩⟩ ≡
4594 \chardef\bbl@bidimode\z@
4595 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4596 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4597 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4598 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4599 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4600 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4601 ⟨⟨/More package options⟩⟩
```

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

```
4602 ⟨⟨*Font selection⟩⟩ ≡
4603 \bbl@trace{Font handling with fontspec}
4604 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4605 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4606 \DisableBabelHook{babel-fontspec}
4607 \@onlypreamble\babelfont
4608 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4609   \bbl@foreach{#1}{%
4610     \expandafter\ifx\csname date##1\endcsname\relax
4611       \IfFileExists{babel-##1.tex}%
4612         {\babelprovide{##1}}%
4613         {}%
4614     \fi}%
4615   \edef\bbl@tempa{#1}%
4616   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4617   \ifx\fontspec\@undefined
4618     \usepackage{fontspec}%
4619   \fi
4620   \EnableBabelHook{babel-fontspec}%
4621   \bbl@bblfont}
4622 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4623   \bbl@ifunset{\bbl@tempb family}%
4624     {\bbl@providefam{\bbl@tempb}}%
4625     {}%
4626   % For the default font, just in case:
4627   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4628   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4629     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4630      \bbl@exp{%
4631        \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4632        \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4633                \<\bbl@tempb default>\<\bbl@tempb family>}}%
4634     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4635        \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:

```
4636 \def\bbl@providefam#1{%
4637   \bbl@exp{%
4638     \\\newcommand\<#1default>{}% Just define it
4639     \\\bbl@add@list\\\bbl@font@fams{#1}%
4640     \\\DeclareRobustCommand\<#1family>{%
4641       \\\not@math@alphabet\<#1family>\relax
4642       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4643       \\\fontfamily\<#1default>%
4644       \<\ifx>\\\UseHooks\\\@undefined\<\else>\\\UseHook{#1family}\<\fi>%
4645       \\\selectfont}%
4646     \\\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.

```
4647 \def\bbl@nostdfont#1{%
4648   \bbl@ifunset{bbl@WFF@\f@family}%
4649     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4650      \bbl@infowarn{The current font is not a babel standard family:\\%
4651        #1%
4652        \fontname\font\\%
4653        There is nothing intrinsically wrong with this warning, and\\%
4654        you can ignore it altogether if you do not need these\\%
4655        families. But if they are used in the document, you should be\\%
4656        aware 'babel' will not set Script and Language for them, so\\%
```

```
4657        you may consider defining a new family with \string\babelfont.\\%
4658        See the manual for further details about \string\babelfont.\\%
4659        Reported}}
4660    {}}}%
4661 \gdef\bbl@switchfont{%
4662   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4663   \bbl@exp{%  eg Arabic -> arabic
4664     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4665   \bbl@foreach\bbl@font@fams{%
4666     \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
4667       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%      (2) from script?
4668         {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4669           {}%                                    123=F - nothing!
4670           {\bbl@exp{%                            3=T - from generic
4671             \global\let\<bbl@##1dflt@\languagename>%
4672                       \<bbl@##1dflt@>}}}%
4673         {\bbl@exp{%                              2=T - from script
4674           \global\let\<bbl@##1dflt@\languagename>%
4675                     \<bbl@##1dflt@*\bbl@tempa>}}}%
4676       {}}%                                       1=T - language, already defined
4677 \def\bbl@tempa{\bbl@nostdfont{}}%  TODO. Don't use \bbl@tempa
4678 \bbl@foreach\bbl@font@fams{%     don't gather with prev for
4679   \bbl@ifunset{bbl@##1dflt@\languagename}%
4680     {\bbl@cs{famrst@##1}%
4681      \global\bbl@csarg\let{famrst@##1}\relax}%
4682     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4683         \\\bbl@add\\\originalTeX{%
4684           \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4685                     \<##1default>\<##1family>{##1}}%
4686         \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4687                   \<##1default>\<##1family>}}}%
4688 \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.

```
4689 \ifx\f@family\@undefined\else   % if latex
4690   \ifcase\bbl@engine            % if pdftex
4691     \let\bbl@ckeckstdfonts\relax
4692   \else
4693     \def\bbl@ckeckstdfonts{%
4694       \begingroup
4695         \global\let\bbl@ckeckstdfonts\relax
4696         \let\bbl@tempa\@empty
4697         \bbl@foreach\bbl@font@fams{%
4698           \bbl@ifunset{bbl@##1dflt@}%
4699             {\@nameuse{##1family}%
4700              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4701              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4702                \space\space\fontname\font\\\\}}%
4703              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4704              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4705             {}}%
4706         \ifx\bbl@tempa\@empty\else
4707           \bbl@infowarn{The following font families will use the default\\%
4708             settings for all or some languages:\\%
4709             \bbl@tempa
4710             There is nothing intrinsically wrong with it, but\\%
4711             'babel' will no set Script and Language, which could\\%
4712             be relevant in some languages. If your document uses\\%
4713             these families, consider redefining them with \string\babelfont.\\%
4714             Reported}%
4715         \fi
4716       \endgroup}
```

```
4717    \fi
4718 \fi
```

Now the macros defining the font with fontspec.

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

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

```
4719 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4720   \bbl@xin@{<>}{#1}%
4721   \ifin@
4722     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4723   \fi
4724 \bbl@exp{%              'Unprotected' macros return prev values
4725    \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4726    \\\bbl@ifsamestring{#2}{\f@family}%
4727      {\\#3%
4728       \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4729       \let\\\bbl@tempa\relax}%
4730      {}}}
4731 %     TODO - next should be global?, but even local does its job. I'm
4732 %     still not sure -- must investigate:
4733 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4734   \let\bbl@tempe\bbl@mapselect
4735   \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4736   \bbl@exp{\\\bbl@replace\\\bbl@tempb{\bbl@stripslash\family/}{}}%
4737   \let\bbl@mapselect\relax
4738   \let\bbl@temp@fam#4%       eg, '\rmfamily', to be restored below
4739   \let#4\@empty      %       Make sure \renewfontfamily is valid
4740   \bbl@exp{%
4741     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4742     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4743       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4744     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4745       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4746     \\\renewfontfamily\\#4%
4747       [\bbl@cl{lsys},% xetex removes unknown features :-(
4748        \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4749        #2]}{#3}% ie \bbl@exp{..}{#3}
4750   \begingroup
4751      #4%
4752      \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4753   \endgroup % TODO. Find better tests:
4754   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4755     {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4756   \ifin@
4757     \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4758   \fi
4759   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4760     {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4761   \ifin@
4762     \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4763   \fi
4764   \let#4\bbl@temp@fam
4765   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4766   \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.

```
4767 \def\bbl@font@rst#1#2#3#4{%
4768   \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.

```
4769 \def\bbl@font@fams{rm,sf,tt}
4770 ⟨⟨/Font selection⟩⟩
```

\BabelFootnote   Footnotes.

```
4771 ⟨⟨*Footnote changes⟩⟩ ≡
4772 \bbl@trace{Bidi footnotes}
4773 \ifnum\bbl@bidimode>\z@ % Any bidi=
4774   \def\bbl@footnote#1#2#3{%
4775     \@ifnextchar[%
4776       {\bbl@footnote@o{#1}{#2}{#3}}%
4777       {\bbl@footnote@x{#1}{#2}{#3}}}
4778   \long\def\bbl@footnote@x#1#2#3#4{%
4779     \bgroup
4780       \select@language@x{\bbl@main@language}%
4781       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4782     \egroup}
4783   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4784     \bgroup
4785       \select@language@x{\bbl@main@language}%
4786       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4787     \egroup}
4788   \def\bbl@footnotetext#1#2#3{%
4789     \@ifnextchar[%
4790       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4791       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4792   \long\def\bbl@footnotetext@x#1#2#3#4{%
4793     \bgroup
4794       \select@language@x{\bbl@main@language}%
4795       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4796     \egroup}
4797   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4798     \bgroup
4799       \select@language@x{\bbl@main@language}%
4800       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4801     \egroup}
4802   \def\BabelFootnote#1#2#3#4{%
4803     \ifx\bbl@fn@footnote\@undefined
4804       \let\bbl@fn@footnote\footnote
4805     \fi
4806     \ifx\bbl@fn@footnotetext\@undefined
4807       \let\bbl@fn@footnotetext\footnotetext
4808     \fi
4809     \bbl@ifblank{#2}%
4810       {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4811        \@namedef{\bbl@stripslash#1text}%
4812          {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4813       {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4814        \@namedef{\bbl@stripslash#1text}%
4815          {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4816 \fi
4817 ⟨⟨/Footnote changes⟩⟩
```

10. Hooks for XeTeX and LuaTeX

10.1. XeTeX

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

Now, the code.

```
4818 ⟨∗xetex⟩
4819 \def\BabelStringsDefault{unicode}
4820 \let\xebbl@stop\relax
4821 \AddBabelHook{xetex}{encodedcommands}{%
4822   \def\bbl@tempa{#1}%
4823   \ifx\bbl@tempa\@empty
4824     \XeTeXinputencoding"bytes"%
4825   \else
4826     \XeTeXinputencoding"#1"%
4827   \fi
4828   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4829 \AddBabelHook{xetex}{stopcommands}{%
4830   \xebbl@stop
4831   \let\xebbl@stop\relax}
4832 \def\bbl@input@classes{% Used in CJK intraspaces
4833   \input{load-unicode-xetex-classes.tex}%
4834   \let\bbl@input@classes\relax}
4835 \def\bbl@intraspace#1 #2 #3\@@{%
4836   \bbl@csarg\gdef{xeisp@\languagename}%
4837     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4838 \def\bbl@intrapenalty#1\@@{%
4839   \bbl@csarg\gdef{xeipn@\languagename}%
4840     {\XeTeXlinebreakpenalty #1\relax}}
4841 \def\bbl@provide@intraspace{%
4842   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4843   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4844   \ifin@
4845     \bbl@ifunset{bbl@intsp@\languagename}{}%
4846       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4847         \ifx\bbl@KVP@intraspace\@nnil
4848           \bbl@exp{%
4849             \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4850         \fi
4851         \ifx\bbl@KVP@intrapenalty\@nnil
4852           \bbl@intrapenalty0\@@
4853         \fi
4854       \fi
4855     \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4856       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4857     \fi
4858     \ifx\bbl@KVP@intrapenalty\@nnil\else
4859       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4860     \fi
4861     \bbl@exp{%
4862       % TODO. Execute only once (but redundant):
4863       \\\bbl@add\<extras\languagename>{%
4864         \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4865         \<bbl@xeisp@\languagename>%
4866         \<bbl@xeipn@\languagename>}%
4867       \\\bbl@toglobal\<extras\languagename>%
4868       \\\bbl@add\<noextras\languagename>{%
4869         \XeTeXlinebreaklocale ""}%
4870       \\\bbl@toglobal\<noextras\languagename>}%
4871     \ifx\bbl@ispacesize\@undefined
4872       \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4873       \ifx\AtBeginDocument\@notprerr
```

```
4874        \expandafter\@secondoftwo  % to execute right now
4875      \fi
4876      \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4877    \fi}%
4878  \fi}
4879 \ifx\DisableBabelHook\@undefined\endinput\fi %%%% TODO: why
4880 <@Font selection@>
4881 \def\bbl@provide@extra#1{}
```

10.2. Support for interchar

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

```
4882 \ifnum\xe@alloc@intercharclass<\thr@@
4883   \xe@alloc@intercharclass\thr@@
4884 \fi
4885 \chardef\bbl@xeclass@default@=\z@
4886 \chardef\bbl@xeclass@cjkideogram@=\@ne
4887 \chardef\bbl@xeclass@cjkleftpunctuation@=\tw@
4888 \chardef\bbl@xeclass@cjkrightpunctuation@=\thr@@
4889 \chardef\bbl@xeclass@boundary@=4095
4890 \chardef\bbl@xeclass@ignore@=4096
```

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

```
4891 \AddBabelHook{babel-interchar}{beforeextras}{%
4892   \@nameuse{bbl@xechars@\languagename}}
4893 \DisableBabelHook{babel-interchar}
4894 \protected\def\bbl@charclass#1{%
4895   \ifnum\count@<\z@
4896     \count@-\count@
4897     \loop
4898       \bbl@exp{%
4899         \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4900       \XeTeXcharclass\count@ \bbl@tempc
4901       \ifnum\count@<`#1\relax
4902       \advance\count@\@ne
4903     \repeat
4904   \else
4905     \babel@savevariable{\XeTeXcharclass`#1}%
4906     \XeTeXcharclass`#1 \bbl@tempc
4907   \fi
4908   \count@`#1\relax}
```

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

```
4909 \newcommand\bbl@ifinterchar[1]{%
4910   \let\bbl@tempa\@gobble        % Assume to ignore
4911   \edef\bbl@tempb{\zap@space#1 \@empty}%
4912   \ifx\bbl@KVP@interchar\@nnil\else
4913     \bbl@replace\bbl@KVP@interchar{ }{,}%
4914     \bbl@foreach\bbl@tempb{%
4915       \bbl@xin@{,##1,}{,\bbl@KVP@interchar,}%
4916       \ifin@
4917         \let\bbl@tempa\@firstofone
4918       \fi}%
4919   \fi
```

```
4920    \bbl@tempa}
4921 \newcommand\IfBabelIntercharT[2]{%
4922    \bbl@carg\bbl@add{bbl@icsave@\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
4923 \newcommand\babelcharclass[3]{%
4924    \EnableBabelHook{babel-interchar}%
4925    \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
4926    \def\bbl@tempb##1{%
4927      \ifx##1\@empty\else
4928        \ifx##1-%
4929          \bbl@upto
4930        \else
4931          \bbl@charclass{%
4932            \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4933        \fi
4934        \expandafter\bbl@tempb
4935      \fi}%
4936    \bbl@ifunset{bbl@xechars@#1}%
4937      {\toks@{%
4938        \babel@savevariable\XeTeXintercharaxtokenstate
4939        \XeTeXintercharaxtokenstate\@ne
4940      }}%
4941      {\toks@\expandafter\expandafter\expandafter{%
4942        \csname bbl@xechars@#1\endcsname}}%
4943    \bbl@csarg\edef{xechars@#1}{%
4944      \the\toks@
4945      \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
4946      \bbl@tempb#3\@empty}}%
4947 \protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
4948 \protected\def\bbl@upto{%
4949    \ifnum\count@>\z@
4950      \advance\count@\@ne
4951      \count@-\count@
4952    \else\ifnum\count@=\z@
4953      \bbl@charclass{-}%
4954    \else
4955      \bbl@error{double-hyphens-class}{}{}{}%
4956    \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⟩.

```
4957 \def\bbl@ignoreinterchar{%
4958    \ifnum\language=\l@nohyphenation
4959      \expandafter\@gobble
4960    \else
4961      \expandafter\@firstofone
4962    \fi}
4963 \newcommand\babelinterchar[5][]{%
4964    \let\bbl@kv@label\@empty
4965    \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
4966    \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4967      {\bbl@ignoreinterchar{#5}}%
4968    \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
4969    \bbl@exp{\\\bbl@for\\\bbl@tempa{\zap@space#3 \@empty}}{%
4970      \bbl@exp{\\\bbl@for\\\bbl@tempb{\zap@space#4 \@empty}}{%
4971        \XeTeXintercharaxtoks
4972          \@nameuse{bbl@xeclass@\bbl@tempa @%
4973            \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{}{#2}} %
4974          \@nameuse{bbl@xeclass@\bbl@tempb @%
4975            \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{}{#2}} %
4976        = \expandafter{%
4977            \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
4978            \csname\zap@space bbl@xeinter@\bbl@kv@label
```

```
4979              @#3@#4@#2 \@empty\endcsname}}}}
4980 \DeclareRobustCommand\enablelocaleinterchar[1]{%
4981   \bbl@ifunset{bbl@ic@#1@\languagename}%
4982     {\bbl@error{unknown-interchar}{#1}{}{}}%
4983     {\bbl@csarg\let{ic@#1@\languagename}\@firstofone}}
4984 \DeclareRobustCommand\disablelocaleinterchar[1]{%
4985   \bbl@ifunset{bbl@ic@#1@\languagename}%
4986     {\bbl@error{unknown-interchar-b}{#1}{}{}}%
4987     {\bbl@csarg\let{ic@#1@\languagename}\@gobble}}
4988 ⟨/xetex⟩
```

10.3. Layout

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

```
4989 ⟨∗xetex | texxet⟩
4990 \providecommand\bbl@provide@intraspace{}
4991 \bbl@trace{Redefinitions for bidi layout}
4992 \def\bbl@sspre@caption{%  TODO: Unused!
4993   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}}
4994 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4995 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4996 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4997 \ifnum\bbl@bidimode>\z@  % TODO: always?
4998   \def\@hangfrom#1{%
4999     \setbox\@tempboxa\hbox{{#1}}%
5000     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5001     \noindent\box\@tempboxa}
5002   \def\raggedright{%
5003     \let\\\@centercr
5004     \bbl@startskip\z@skip
5005     \@rightskip\@flushglue
5006     \bbl@endskip\@rightskip
5007     \parindent\z@
5008     \parfillskip\bbl@startskip}
5009   \def\raggedleft{%
5010     \let\\\@centercr
5011     \bbl@startskip\@flushglue
5012     \bbl@endskip\z@skip
5013     \parindent\z@
5014     \parfillskip\bbl@endskip}
5015 \fi
5016 \IfBabelLayout{lists}
5017   {\bbl@sreplace\list
5018     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5019   \def\bbl@listleftmargin{%
5020     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5021   \ifcase\bbl@engine
5022     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
5023     \def\p@enumiii{\p@enumii)\theenumii(}%
5024   \fi
5025   \bbl@sreplace\@verbatim
5026     {\leftskip\@totalleftmargin}%
5027     {\bbl@startskip\textwidth
5028      \advance\bbl@startskip-\linewidth}%
5029   \bbl@sreplace\@verbatim
5030     {\rightskip\z@skip}%
5031     {\bbl@endskip\z@skip}}%
```

```
5032    {}
5033 \IfBabelLayout{contents}
5034    {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5035     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5036    {}
5037 \IfBabelLayout{columns}
5038    {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
5039     \def\bbl@outputhbox#1{%
5040        \hb@xt@\textwidth{%
5041           \hskip\columnwidth
5042           \hfil
5043           {\normalcolor\vrule \@width\columnseprule}%
5044           \hfil
5045           \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5046           \hskip-\textwidth
5047           \hb@xt@\columnwidth{\box\@outputbox \hss}%
5048           \hskip\columnsep
5049           \hskip\columnwidth}}}%
5050    {}
5051 <@Footnote changes@>
5052 \IfBabelLayout{footnotes}%
5053    {\BabelFootnote\footnote\languagename{}{}%
5054     \BabelFootnote\localfootnote\languagename{}{}%
5055     \BabelFootnote\mainfootnote{}{}{}}
5056    {}
```

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.

```
5057 \IfBabelLayout{counters*}%
5058    {\bbl@add\bbl@opt@layout{.counters.}%
5059     \AddToHook{shipout/before}{%
5060        \let\bbl@tempa\babelsublr
5061        \let\babelsublr\@firstofone
5062        \let\bbl@save@thepage\thepage
5063        \protected@edef\thepage{\thepage}%
5064        \let\babelsublr\bbl@tempa}%
5065     \AddToHook{shipout/after}{%
5066        \let\thepage\bbl@save@thepage}}{}
5067 \IfBabelLayout{counters}%
5068    {\let\bbl@latinarabic=\@arabic
5069     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5070     \let\bbl@asciiroman=\@roman
5071     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
5072     \let\bbl@asciiRoman=\@Roman
5073     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5074 \fi % end if layout
5075 ⟨/xetex | texxet⟩
```

10.4. 8-bit TeX

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

```
5076 ⟨*texxet⟩
5077 \def\bbl@provide@extra#1{%
5078  % == auto-select encoding ==
5079  \ifx\bbl@encoding@select@off\@empty\else
5080     \bbl@ifunset{bbl@encoding@#1}%
5081        {\def\@elt##1{,##1,}%
5082         \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5083         \count@\z@
5084         \bbl@foreach\bbl@tempe{%
5085            \def\bbl@tempd{##1}%  Save last declared
5086            \advance\count@\@ne}%
```

```
5087        \ifnum\count@>\@ne    % (1)
5088          \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5089          \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5090          \bbl@replace\bbl@tempa{ }{,}%
5091          \global\bbl@csarg\let{encoding@#1}\@empty
5092          \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5093          \ifin@\else % if main encoding included in ini, do nothing
5094            \let\bbl@tempb\relax
5095            \bbl@foreach\bbl@tempa{%
5096              \ifx\bbl@tempb\relax
5097                \bbl@xin@{,##1,}{,\bbl@tempe,}%
5098                \ifin@\def\bbl@tempb{##1}\fi
5099              \fi}%
5100            \ifx\bbl@tempb\relax\else
5101              \bbl@exp{%
5102                \global\<bbl@add>\<bbl@preextras@#1>{\<bbl@encoding@#1>}%
5103              \gdef\<bbl@encoding@#1>{%
5104                \\\babel@save\\\f@encoding
5105                \\\bbl@add\\\originalTeX{\\\selectfont}%
5106                \\\fontencoding{\bbl@tempb}%
5107                \\\selectfont}}%
5108            \fi
5109          \fi
5110        \fi}%
5111      {}%
5112  \fi}
5113 ⟨/texxet⟩
```

10.5. LuaTeX

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

The names `\l@⟨language⟩` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\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, the are added, but note if the language patterns have not been preloaded they won't at run time.

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

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

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

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

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

```
5114 ⟨*luatex⟩
```

```
5115 \directlua{ Babel = Babel or {} } % DL2
5116 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
5117 \bbl@trace{Read language.dat}
5118 \ifx\bbl@readstream\@undefined
5119   \csname newread\endcsname\bbl@readstream
5120 \fi
5121 \begingroup
5122   \toks@{}
5123   \count@\z@ % 0=start, 1=0th, 2=normal
5124   \def\bbl@process@line#1#2 #3 #4 {%
5125     \ifx=#1%
5126       \bbl@process@synonym{#2}%
5127     \else
5128       \bbl@process@language{#1#2}{#3}{#4}%
5129     \fi
5130     \ignorespaces}
5131   \def\bbl@manylang{%
5132     \ifnum\bbl@last>\@ne
5133       \bbl@info{Non-standard hyphenation setup}%
5134     \fi
5135     \let\bbl@manylang\relax}
5136   \def\bbl@process@language#1#2#3{%
5137     \ifcase\count@
5138       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5139     \or
5140       \count@\tw@
5141     \fi
5142     \ifnum\count@=\tw@
5143       \expandafter\addlanguage\csname l@#1\endcsname
5144       \language\allocationnumber
5145       \chardef\bbl@last\allocationnumber
5146       \bbl@manylang
5147       \let\bbl@elt\relax
5148       \xdef\bbl@languages{%
5149         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5150     \fi
5151     \the\toks@
5152     \toks@{}}
5153   \def\bbl@process@synonym@aux#1#2{%
5154     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5155     \let\bbl@elt\relax
5156     \xdef\bbl@languages{%
5157       \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
5158   \def\bbl@process@synonym#1{%
5159     \ifcase\count@
5160       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5161     \or
5162       \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5163     \else
5164       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5165     \fi}
5166 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
5167   \chardef\l@english\z@
5168   \chardef\l@USenglish\z@
5169   \chardef\bbl@last\z@
5170   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
5171   \gdef\bbl@languages{%
5172     \bbl@elt{english}{0}{hyphen.tex}{}%
5173     \bbl@elt{USenglish}{0}{}{}}
5174 \else
5175   \global\let\bbl@languages@format\bbl@languages
5176   \def\bbl@elt#1#2#3#4{% Remove all except language 0
5177     \ifnum#2>\z@\else
```

111

```
5178          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5179        \fi}%
5180      \xdef\bbl@languages{\bbl@languages}%
5181    \fi
5182    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
5183    \bbl@languages
5184    \openin\bbl@readstream=language.dat
5185    \ifeof\bbl@readstream
5186      \bbl@warning{I couldn't find language.dat. No additional\\%
5187                   patterns loaded. Reported}%
5188    \else
5189      \loop
5190        \endlinechar\m@ne
5191        \read\bbl@readstream to \bbl@line
5192        \endlinechar`\^^M
5193        \if T\ifeof\bbl@readstream F\fi T\relax
5194          \ifx\bbl@line\@empty\else
5195            \edef\bbl@line{\bbl@line\space\space\space}%
5196            \expandafter\bbl@process@line\bbl@line\relax
5197          \fi
5198      \repeat
5199    \fi
5200    \closein\bbl@readstream
5201  \endgroup
5202  \bbl@trace{Macros for reading patterns files}
5203  \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
5204  \ifx\babelcatcodetablenum\@undefined
5205    \ifx\newcatcodetable\@undefined
5206      \def\babelcatcodetablenum{5211}
5207      \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5208    \else
5209      \newcatcodetable\babelcatcodetablenum
5210      \newcatcodetable\bbl@pattcodes
5211    \fi
5212  \else
5213    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5214  \fi
5215  \def\bbl@luapatterns#1#2{%
5216    \bbl@get@enc#1::\@@@
5217    \setbox\z@\hbox\bgroup
5218      \begingroup
5219        \savecatcodetable\babelcatcodetablenum\relax
5220        \initcatcodetable\bbl@pattcodes\relax
5221        \catcodetable\bbl@pattcodes\relax
5222          \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5223          \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5224          \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5225          \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5226          \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5227          \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
5228          \input #1\relax
5229        \catcodetable\babelcatcodetablenum\relax
5230      \endgroup
5231      \def\bbl@tempa{#2}%
5232      \ifx\bbl@tempa\@empty\else
5233        \input #2\relax
5234      \fi
5235    \egroup}%
5236  \def\bbl@patterns@lua#1{%
5237    \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5238      \csname l@#1\endcsname
5239      \edef\bbl@tempa{#1}%
5240    \else
```

112

```
5241      \csname l@#1:\f@encoding\endcsname
5242      \edef\bbl@tempa{#1:\f@encoding}%
5243  \fi\relax
5244  \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
5245  \@ifundefined{bbl@hyphendata@\the\language}%
5246      {\def\bbl@elt##1##2##3##4{%
5247          \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5248            \def\bbl@tempb{##3}%
5249            \ifx\bbl@tempb\@empty\else % if not a synonymous
5250              \def\bbl@tempc{{##3}{##4}}%
5251            \fi
5252            \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5253          \fi}%
5254        \bbl@languages
5255        \@ifundefined{bbl@hyphendata@\the\language}%
5256          {\bbl@info{No hyphenation patterns were set for\\%
5257                    language '\bbl@tempa'. Reported}}%
5258          {\expandafter\expandafter\expandafter\bbl@luapatterns
5259            \csname bbl@hyphendata@\the\language\endcsname}}{}}
5260  \endinput\fi
```

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

```
5261  \ifx\DisableBabelHook\@undefined
5262    \AddBabelHook{luatex}{everylanguage}{%
5263      \def\process@language##1##2##3{%
5264        \def\process@line####1####2 ####3 ####4 {}}}
5265    \AddBabelHook{luatex}{loadpatterns}{%
5266      \input #1\relax
5267      \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
5268        {{#1}{}}}
5269    \AddBabelHook{luatex}{loadexceptions}{%
5270      \input #1\relax
5271      \def\bbl@tempb##1##2{{##1}{#1}}%
5272      \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
5273        {\expandafter\expandafter\expandafter\bbl@tempb
5274         \csname bbl@hyphendata@\the\language\endcsname}}
5275  \endinput\fi
```

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

```
5276  \begingroup  % TODO - to a lua file % DL3
5277  \catcode`\%=12
5278  \catcode`\'=12
5279  \catcode`\"=12
5280  \catcode`\:=12
5281  \directlua{
5282    Babel.locale_props = Babel.locale_props or {}
5283    function Babel.lua_error(e, a)
5284      tex.print([[\noexpand\csname bbl@error\endcsname{]] ..
5285        e .. '}{' .. (a or '') .. '}{}{}')
5286    end
5287    function Babel.bytes(line)
5288      return line:gsub("(.)",
5289        function (chr) return unicode.utf8.char(string.byte(chr)) end)
5290    end
5291    function Babel.begin_process_input()
5292      if luatexbase and luatexbase.add_to_callback then
5293        luatexbase.add_to_callback('process_input_buffer',
5294                                    Babel.bytes,'Babel.bytes')
5295      else
5296        Babel.callback = callback.find('process_input_buffer')
5297        callback.register('process_input_buffer',Babel.bytes)
5298      end
5299    end
```

```lua
function Babel.end_process_input ()
  if luatexbase and luatexbase.remove_from_callback then
    luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
  else
    callback.register('process_input_buffer',Babel.callback)
  end
end
function Babel.str_to_nodes(fn, matches, base)
  local n, head, last
  if fn == nil then return nil end
  for s in string.utfvalues(fn(matches)) do
    if base.id == 7 then
      base = base.replace
    end
    n = node.copy(base)
    n.char   = s
    if not head then
      head = n
    else
      last.next = n
    end
    last = n
  end
  return head
end
Babel.linebreaking = Babel.linebreaking or {}
Babel.linebreaking.before = {}
Babel.linebreaking.after = {}
Babel.locale = {}
function Babel.linebreaking.add_before(func, pos)
  tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
  if pos == nil then
    table.insert(Babel.linebreaking.before, func)
  else
    table.insert(Babel.linebreaking.before, pos, func)
  end
end
function Babel.linebreaking.add_after(func)
  tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
  table.insert(Babel.linebreaking.after, func)
end
function Babel.addpatterns(pp, lg)
  local lg = lang.new(lg)
  local pats = lang.patterns(lg) or ''
  lang.clear_patterns(lg)
  for p in pp:gmatch('[^%s]+') do
    ss = ''
    for i in string.utfcharacters(p:gsub('%d', '')) do
      ss = ss .. '%d?' .. i
    end
    ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
    ss = ss:gsub('%.%%d%?$', '%%.')
    pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
    if n == 0 then
      tex.sprint(
        [[\string\csname\space bbl@info\endcsname{New pattern: ]]
        .. p .. [[}]])
      pats = pats .. ' ' .. p
    else
      tex.sprint(
        [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
        .. p .. [[}]])
    end
```

```
5363        end
5364      lang.patterns(lg, pats)
5365    end
5366    Babel.characters = Babel.characters or {}
5367    Babel.ranges = Babel.ranges or {}
5368    function Babel.hlist_has_bidi(head)
5369      local has_bidi = false
5370      local ranges = Babel.ranges
5371      for item in node.traverse(head) do
5372        if item.id == node.id'glyph' then
5373          local itemchar = item.char
5374          local chardata = Babel.characters[itemchar]
5375          local dir = chardata and chardata.d or nil
5376          if not dir then
5377            for nn, et in ipairs(ranges) do
5378              if itemchar < et[1] then
5379                break
5380              elseif itemchar <= et[2] then
5381                dir = et[3]
5382                break
5383              end
5384            end
5385          end
5386          if dir and (dir == 'al' or dir == 'r') then
5387            has_bidi = true
5388          end
5389        end
5390      end
5391      return has_bidi
5392    end
5393    function Babel.set_chranges_b (script, chrng)
5394      if chrng == '' then return end
5395      texio.write('Replacing ' .. script .. ' script ranges')
5396      Babel.script_blocks[script] = {}
5397      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5398        table.insert(
5399          Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5400      end
5401    end
5402    function Babel.discard_sublr(str)
5403      if str:find( [[\string\indexentry]] ) and
5404           str:find( [[\string\babelsublr]] ) then
5405        str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5406                        function(m) return m:sub(2,-2) end )
5407      end
5408      return str
5409    end
5410  }
5411  \endgroup
5412  \ifx\newattribute\@undefined\else % Test for plain
5413    \newattribute\bbl@attr@locale % DL4
5414    \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5415    \AddBabelHook{luatex}{beforeextras}{%
5416      \setattribute\bbl@attr@locale\localeid}
5417  \fi
5418  \def\BabelStringsDefault{unicode}
5419  \let\luabbl@stop\relax
5420  \AddBabelHook{luatex}{encodedcommands}{%
5421    \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5422    \ifx\bbl@tempa\bbl@tempb\else
5423      \directlua{Babel.begin_process_input()}%
5424      \def\luabbl@stop{%
5425        \directlua{Babel.end_process_input()}}%
```

```
5426    \fi}%
5427 \AddBabelHook{luatex}{stopcommands}{%
5428    \luabbl@stop
5429    \let\luabbl@stop\relax}
5430 \AddBabelHook{luatex}{patterns}{%
5431    \@ifundefined{bbl@hyphendata@\the\language}%
5432      {\def\bbl@elt##1##2##3##4{%
5433        \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5434          \def\bbl@tempb{##3}%
5435          \ifx\bbl@tempb\@empty\else % if not a synonymous
5436            \def\bbl@tempc{{##3}{##4}}%
5437          \fi
5438          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5439        \fi}%
5440       \bbl@languages
5441       \@ifundefined{bbl@hyphendata@\the\language}%
5442        {\bbl@info{No hyphenation patterns were set for\\%
5443                  language '#2'. Reported}}%
5444        {\expandafter\expandafter\expandafter\bbl@luapatterns
5445          \csname bbl@hyphendata@\the\language\endcsname}}{}%
5446    \@ifundefined{bbl@patterns@}{}{%
5447      \begingroup
5448        \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5449        \ifin@\else
5450          \ifx\bbl@patterns@\@empty\else
5451            \directlua{ Babel.addpatterns(
5452              [[\bbl@patterns@]], \number\language) }%
5453          \fi
5454          \@ifundefined{bbl@patterns@#1}%
5455            \@empty
5456            {\directlua{ Babel.addpatterns(
5457                [[\space\csname bbl@patterns@#1\endcsname]],
5458                \number\language) }}%
5459          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5460        \fi
5461      \endgroup}%
5462    \bbl@exp{%
5463      \bbl@ifunset{bbl@prehc@\languagename}{}%
5464        {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5465          {\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.

```
5466 \@onlypreamble\babelpatterns
5467 \AtEndOfPackage{%
5468   \newcommand\babelpatterns[2][\@empty]{%
5469     \ifx\bbl@patterns@\relax
5470       \let\bbl@patterns@\@empty
5471     \fi
5472     \ifx\bbl@pttnlist\@empty\else
5473       \bbl@warning{%
5474         You must not intermingle \string\selectlanguage\space and\\%
5475         \string\babelpatterns\space or some patterns will not\\%
5476         be taken into account. Reported}%
5477     \fi
5478     \ifx\@empty#1%
5479       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5480     \else
5481       \edef\bbl@tempb{\zap@space#1 \@empty}%
5482       \bbl@for\bbl@tempa\bbl@tempb{%
5483         \bbl@fixname\bbl@tempa
5484         \bbl@iflanguage\bbl@tempa{%
```

```
5485            \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5486              \@ifundefined{bbl@patterns@\bbl@tempa}%
5487                \@empty
5488                {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5489          #2}}}%
5490      \fi}}
```

10.6. Southeast Asian scripts

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

Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```
5491 \def\bbl@intraspace#1 #2 #3\@@{%
5492   \directlua{
5493     Babel.intraspaces = Babel.intraspaces or {}
5494     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5495       {b = #1, p = #2, m = #3}
5496     Babel.locale_props[\the\localeid].intraspace = %
5497       {b = #1, p = #2, m = #3}
5498   }}
5499 \def\bbl@intrapenalty#1\@@{%
5500   \directlua{
5501     Babel.intrapenalties = Babel.intrapenalties or {}
5502     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5503     Babel.locale_props[\the\localeid].intrapenalty = #1
5504   }}
5505 \begingroup
5506 \catcode`\%=12
5507 \catcode`\&=14
5508 \catcode`\'=12
5509 \catcode`\~=12
5510 \gdef\bbl@seaintraspace{&
5511   \let\bbl@seaintraspace\relax
5512   \directlua{
5513     Babel.sea_enabled = true
5514     Babel.sea_ranges = Babel.sea_ranges or {}
5515     function Babel.set_chranges (script, chrng)
5516       local c = 0
5517       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5518         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5519         c = c + 1
5520       end
5521     end
5522     function Babel.sea_disc_to_space (head)
5523       local sea_ranges = Babel.sea_ranges
5524       local last_char = nil
5525       local quad = 655360        &% 10 pt = 655360 = 10 * 65536
5526       for item in node.traverse(head) do
5527         local i = item.id
5528         if i == node.id'glyph' then
5529           last_char = item
5530         elseif i == 7 and item.subtype == 3 and last_char
5531             and last_char.char > 0x0C99 then
5532           quad = font.getfont(last_char.font).size
5533           for lg, rg in pairs(sea_ranges) do
5534             if last_char.char > rg[1] and last_char.char < rg[2] then
5535               lg = lg:sub(1, 4)  &% Remove trailing number of, eg, Cyrl1
5536               local intraspace = Babel.intraspaces[lg]
5537               local intrapenalty = Babel.intrapenalties[lg]
5538               local n
5539               if intrapenalty ~= 0 then
5540                 n = node.new(14, 0)      &% penalty
```

```
5541              n.penalty = intrapenalty
5542              node.insert_before(head, item, n)
5543            end
5544            n = node.new(12, 13)       &% (glue, spaceskip)
5545            node.setglue(n, intraspace.b * quad,
5546                            intraspace.p * quad,
5547                            intraspace.m * quad)
5548            node.insert_before(head, item, n)
5549            node.remove(head, item)
5550          end
5551        end
5552      end
5553    end
5554  end
5555 }&
5556 \bbl@luahyphenate}
```

10.7. CJK line breaking

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

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

```
5557 \catcode`\%=14
5558 \gdef\bbl@cjkintraspace{%
5559  \let\bbl@cjkintraspace\relax
5560  \directlua{
5561    require('babel-data-cjk.lua')
5562    Babel.cjk_enabled = true
5563    function Babel.cjk_linebreak(head)
5564      local GLYPH = node.id'glyph'
5565      local last_char = nil
5566      local quad = 655360       % 10 pt = 655360 = 10 * 65536
5567      local last_class = nil
5568      local last_lang = nil
5569
5570      for item in node.traverse(head) do
5571        if item.id == GLYPH then
5572
5573          local lang = item.lang
5574
5575          local LOCALE = node.get_attribute(item,
5576                Babel.attr_locale)
5577          local props = Babel.locale_props[LOCALE]
5578
5579          local class = Babel.cjk_class[item.char].c
5580
5581          if props.cjk_quotes and props.cjk_quotes[item.char] then
5582            class = props.cjk_quotes[item.char]
5583          end
5584
5585          if class == 'cp' then class = 'cl' % )] as CL
5586          elseif class == 'id' then class = 'I'
5587          elseif class == 'cj' then class = 'I' % loose
5588          end
5589
5590          local br = 0
5591          if class and last_class and Babel.cjk_breaks[last_class][class] then
5592            br = Babel.cjk_breaks[last_class][class]
5593          end
5594
```

```
5595          if br == 1 and props.linebreak == 'c' and
5596              lang ~= \the\l@nohyphenation\space and
5597              last_lang ~= \the\l@nohyphenation then
5598            local intrapenalty = props.intrapenalty
5599            if intrapenalty ~= 0 then
5600              local n = node.new(14, 0)      % penalty
5601              n.penalty = intrapenalty
5602              node.insert_before(head, item, n)
5603            end
5604            local intraspace = props.intraspace
5605            local n = node.new(12, 13)      % (glue, spaceskip)
5606            node.setglue(n, intraspace.b * quad,
5607                            intraspace.p * quad,
5608                            intraspace.m * quad)
5609            node.insert_before(head, item, n)
5610          end
5611
5612          if font.getfont(item.font) then
5613            quad = font.getfont(item.font).size
5614          end
5615          last_class = class
5616          last_lang = lang
5617        else % if penalty, glue or anything else
5618          last_class = nil
5619        end
5620      end
5621      lang.hyphenate(head)
5622    end
5623  }%
5624  \bbl@luahyphenate}
5625 \gdef\bbl@luahyphenate{%
5626  \let\bbl@luahyphenate\relax
5627  \directlua{
5628    luatexbase.add_to_callback('hyphenate',
5629    function (head, tail)
5630      if Babel.linebreaking.before then
5631        for k, func in ipairs(Babel.linebreaking.before)  do
5632          func(head)
5633        end
5634      end
5635      lang.hyphenate(head)
5636      if Babel.cjk_enabled then
5637        Babel.cjk_linebreak(head)
5638      end
5639      if Babel.linebreaking.after then
5640        for k, func in ipairs(Babel.linebreaking.after)  do
5641          func(head)
5642        end
5643      end
5644      if Babel.sea_enabled then
5645        Babel.sea_disc_to_space(head)
5646      end
5647    end,
5648    'Babel.hyphenate')
5649  }
5650 }
5651 \endgroup
5652 \def\bbl@provide@intraspace{%
5653  \bbl@ifunset{bbl@intsp@\languagename}{}%
5654    {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5655      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5656      \ifin@              % cjk
5657        \bbl@cjkintraspace
```

```
5658        \directlua{
5659            Babel.locale_props = Babel.locale_props or {}
5660            Babel.locale_props[\the\localeid].linebreak = 'c'
5661        }%
5662        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5663        \ifx\bbl@KVP@intrapenalty\@nnil
5664          \bbl@intrapenalty0\@@
5665        \fi
5666      \else              % sea
5667        \bbl@seaintraspace
5668        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5669        \directlua{
5670            Babel.sea_ranges = Babel.sea_ranges or {}
5671            Babel.set_chranges('\bbl@cl{sbcp}',
5672                               '\bbl@cl{chrng}')
5673        }%
5674        \ifx\bbl@KVP@intrapenalty\@nnil
5675          \bbl@intrapenalty0\@@
5676        \fi
5677      \fi
5678    \fi
5679    \ifx\bbl@KVP@intrapenalty\@nnil\else
5680      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5681    \fi}}
```

10.8. Arabic justification

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

```
5682 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5683 \def\bblar@chars{%
5684   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5685   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5686   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5687 \def\bblar@elongated{%
5688   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5689   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5690   0649,064A}
5691 \begingroup
5692   \catcode`\_=11 \catcode`:=11
5693   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5694 \endgroup
5695 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5696   \let\bbl@arabicjust\relax
5697   \newattribute\bblar@kashida
5698   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5699   \bblar@kashida=\z@
5700   \bbl@patchfont{{\bbl@parsejalt}}%
5701   \directlua{
5702     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5703     Babel.arabic.elong_map[\the\localeid]   = {}
5704     luatexbase.add_to_callback('post_linebreak_filter',
5705       Babel.arabic.justify, 'Babel.arabic.justify')
5706     luatexbase.add_to_callback('hpack_filter',
5707       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5708 }}%
```

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

```
5709 \def\bblar@fetchjalt#1#2#3#4{%
5710 \bbl@exp{\\\bbl@foreach{#1}}{%
5711   \bbl@ifunset{bblar@JE@##1}%
5712     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5713     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
```

```
5714    \directlua{%
5715      local last = nil
5716      for item in node.traverse(tex.box[0].head) do
5717        if item.id == node.id'glyph' and item.char > 0x600 and
5718            not (item.char == 0x200D) then
5719          last = item
5720        end
5721      end
5722      Babel.arabic.#3['##1#4'] = last.char
5723    }}}
```

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

```
5724 \gdef\bbl@parsejalt{%
5725   \ifx\addfontfeature\@undefined\else
5726     \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5727     \ifin@
5728       \directlua{%
5729         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5730           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5731           tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5732         end
5733       }%
5734     \fi
5735   \fi}
5736 \gdef\bbl@parsejalti{%
5737   \begingroup
5738     \let\bbl@parsejalt\relax      % To avoid infinite loop
5739     \edef\bbl@tempb{\fontid\font}%
5740     \bblar@nofswarn
5741     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5742     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5743     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5744     \addfontfeature{RawFeature=+jalt}%
5745   % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5746     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5747     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5748     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5749       \directlua{%
5750         for k, v in pairs(Babel.arabic.from) do
5751           if Babel.arabic.dest[k] and
5752               not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5753             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5754               [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5755           end
5756         end
5757       }%
5758   \endgroup}
```

The actual justification (inspired by CHICKENIZE).

```
5759 \begingroup
5760 \catcode`#=11
5761 \catcode`~=11
5762 \directlua{
5763
5764 Babel.arabic = Babel.arabic or {}
5765 Babel.arabic.from = {}
5766 Babel.arabic.dest = {}
5767 Babel.arabic.justify_factor = 0.95
5768 Babel.arabic.justify_enabled = true
5769 Babel.arabic.kashida_limit = -1
5770
5771 function Babel.arabic.justify(head)
5772   if not Babel.arabic.justify_enabled then return head end
```

121

```
5773  for line in node.traverse_id(node.id'hlist', head) do
5774    Babel.arabic.justify_hlist(head, line)
5775  end
5776  return head
5777 end
5778
5779 function Babel.arabic.justify_hbox(head, gc, size, pack)
5780  local has_inf = false
5781  if Babel.arabic.justify_enabled and pack == 'exactly' then
5782    for n in node.traverse_id(12, head) do
5783      if n.stretch_order > 0 then has_inf = true end
5784    end
5785    if not has_inf then
5786      Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5787    end
5788  end
5789  return head
5790 end
5791
5792 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5793  local d, new
5794  local k_list, k_item, pos_inline
5795  local width, width_new, full, k_curr, wt_pos, goal, shift
5796  local subst_done = false
5797  local elong_map = Babel.arabic.elong_map
5798  local cnt
5799  local last_line
5800  local GLYPH = node.id'glyph'
5801  local KASHIDA = Babel.attr_kashida
5802  local LOCALE = Babel.attr_locale
5803
5804  if line == nil then
5805    line = {}
5806    line.glue_sign = 1
5807    line.glue_order = 0
5808    line.head = head
5809    line.shift = 0
5810    line.width = size
5811  end
5812
5813  % Exclude last line. todo. But-- it discards one-word lines, too!
5814  % ? Look for glue = 12:15
5815  if (line.glue_sign == 1 and line.glue_order == 0) then
5816    elongs = {}     % Stores elongated candidates of each line
5817    k_list = {}     % And all letters with kashida
5818    pos_inline = 0  % Not yet used
5819
5820    for n in node.traverse_id(GLYPH, line.head) do
5821      pos_inline = pos_inline + 1 % To find where it is. Not used.
5822
5823      % Elongated glyphs
5824      if elong_map then
5825        local locale = node.get_attribute(n, LOCALE)
5826        if elong_map[locale] and elong_map[locale][n.font] and
5827            elong_map[locale][n.font][n.char] then
5828          table.insert(elongs, {node = n, locale = locale} )
5829          node.set_attribute(n.prev, KASHIDA, 0)
5830        end
5831      end
5832
5833      % Tatwil
5834      if Babel.kashida_wts then
5835        local k_wt = node.get_attribute(n, KASHIDA)
```

```
5836        if k_wt > 0 then % todo. parameter for multi inserts
5837          table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5838        end
5839      end
5840
5841    end % of node.traverse_id
5842
5843    if #elongs == 0 and #k_list == 0 then goto next_line end
5844    full  = line.width
5845    shift = line.shift
5846    goal  = full * Babel.arabic.justify_factor % A bit crude
5847    width = node.dimensions(line.head)    % The 'natural' width
5848
5849    % == Elongated ==
5850    % Original idea taken from 'chikenize'
5851    while (#elongs > 0 and width < goal) do
5852      subst_done = true
5853      local x = #elongs
5854      local curr = elongs[x].node
5855      local oldchar = curr.char
5856      curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5857      width = node.dimensions(line.head)  % Check if the line is too wide
5858      % Substitute back if the line would be too wide and break:
5859      if width > goal then
5860        curr.char = oldchar
5861        break
5862      end
5863      % If continue, pop the just substituted node from the list:
5864      table.remove(elongs, x)
5865    end
5866
5867    % == Tatwil ==
5868    if #k_list == 0 then goto next_line end
5869
5870    width = node.dimensions(line.head)    % The 'natural' width
5871    k_curr = #k_list % Traverse backwards, from the end
5872    wt_pos = 1
5873
5874    while width < goal do
5875      subst_done = true
5876      k_item = k_list[k_curr].node
5877      if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5878        d = node.copy(k_item)
5879        d.char = 0x0640
5880        d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5881        d.xoffset = 0
5882        line.head, new = node.insert_after(line.head, k_item, d)
5883        width_new = node.dimensions(line.head)
5884        if width > goal or width == width_new then
5885          node.remove(line.head, new) % Better compute before
5886          break
5887        end
5888        if Babel.fix_diacr then
5889          Babel.fix_diacr(k_item.next)
5890        end
5891        width = width_new
5892      end
5893      if k_curr == 1 then
5894        k_curr = #k_list
5895        wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5896      else
5897        k_curr = k_curr - 1
5898      end
```

```
5899     end
5900
5901     % Limit the number of tatweel by removing them. Not very efficient,
5902     % but it does the job in a quite predictable way.
5903     if Babel.arabic.kashida_limit > -1 then
5904       cnt = 0
5905       for n in node.traverse_id(GLYPH, line.head) do
5906         if n.char == 0x0640 then
5907           cnt = cnt + 1
5908           if cnt > Babel.arabic.kashida_limit then
5909             node.remove(line.head, n)
5910           end
5911         else
5912           cnt = 0
5913         end
5914       end
5915     end
5916
5917     ::next_line::
5918
5919     % Must take into account marks and ins, see luatex manual.
5920     % Have to be executed only if there are changes. Investigate
5921     % what's going on exactly.
5922     if subst_done and not gc then
5923       d = node.hpack(line.head, full, 'exactly')
5924       d.shift = shift
5925       node.insert_before(head, line, d)
5926       node.remove(head, line)
5927     end
5928   end % if process line
5929 end
5930 }
5931 \endgroup
5932 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...
```

10.9. Common stuff

```
5933 <@Font selection@>
```

10.10. Automatic fonts and ids switching

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

```
5934 % TODO - to a lua file
5935 \directlua{% DL6
5936 Babel.script_blocks = {
5937   ['dflt'] = {},
5938   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5939              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5940   ['Armn'] = {{0x0530, 0x058F}},
5941   ['Beng'] = {{0x0980, 0x09FF}},
5942   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5943   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5944   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5945              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5946   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5947   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
```

```
5948                {0xAB00, 0xAB2F}},
5949    ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5950    % Don't follow strictly Unicode, which places some Coptic letters in
5951    % the 'Greek and Coptic' block
5952    ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5953    ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5954                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5955                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5956                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5957                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5958                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5959    ['Hebr'] = {{0x0590, 0x05FF}},
5960    ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5961                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5962    ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5963    ['Knda'] = {{0x0C80, 0x0CFF}},
5964    ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5965                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5966                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5967    ['Laoo'] = {{0x0E80, 0x0EFF}},
5968    ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5969                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5970                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5971    ['Mahj'] = {{0x11150, 0x1117F}},
5972    ['Mlym'] = {{0x0D00, 0x0D7F}},
5973    ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5974    ['Orya'] = {{0x0B00, 0x0B7F}},
5975    ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5976    ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5977    ['Taml'] = {{0x0B80, 0x0BFF}},
5978    ['Telu'] = {{0x0C00, 0x0C7F}},
5979    ['Tfng'] = {{0x2D30, 0x2D7F}},
5980    ['Thai'] = {{0x0E00, 0x0E7F}},
5981    ['Tibt'] = {{0x0F00, 0x0FFF}},
5982    ['Vaii'] = {{0xA500, 0xA63F}},
5983    ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5984 }
5985
5986 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5987 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5988 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5989
5990 function Babel.locale_map(head)
5991   if not Babel.locale_mapped then return head end
5992
5993   local LOCALE = Babel.attr_locale
5994   local GLYPH = node.id('glyph')
5995   local inmath = false
5996   local toloc_save
5997   for item in node.traverse(head) do
5998     local toloc
5999     if not inmath and item.id == GLYPH then
6000       % Optimization: build a table with the chars found
6001       if Babel.chr_to_loc[item.char] then
6002         toloc = Babel.chr_to_loc[item.char]
6003       else
6004         for lc, maps in pairs(Babel.loc_to_scr) do
6005           for _, rg in pairs(maps) do
6006             if item.char >= rg[1] and item.char <= rg[2] then
6007               Babel.chr_to_loc[item.char] = lc
6008               toloc = lc
6009               break
6010             end
```

```
6011              end
6012            end
6013          % Treat composite chars in a different fashion, because they
6014          % 'inherit' the previous locale.
6015          if (item.char >= 0x0300 and item.char <= 0x036F) or
6016             (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6017             (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6018              Babel.chr_to_loc[item.char] = -2000
6019              toloc = -2000
6020          end
6021          if not toloc then
6022            Babel.chr_to_loc[item.char] = -1000
6023          end
6024        end
6025        if toloc == -2000 then
6026          toloc = toloc_save
6027        elseif toloc == -1000 then
6028          toloc = nil
6029        end
6030        if toloc and Babel.locale_props[toloc] and
6031            Babel.locale_props[toloc].letters and
6032            tex.getcatcode(item.char) \string~= 11 then
6033          toloc = nil
6034        end
6035        if toloc and Babel.locale_props[toloc].script
6036            and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6037            and Babel.locale_props[toloc].script ==
6038              Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6039          toloc = nil
6040        end
6041        if toloc then
6042          if Babel.locale_props[toloc].lg then
6043            item.lang = Babel.locale_props[toloc].lg
6044            node.set_attribute(item, LOCALE, toloc)
6045          end
6046          if Babel.locale_props[toloc]['/'..item.font] then
6047            item.font = Babel.locale_props[toloc]['/'..item.font]
6048          end
6049        end
6050        toloc_save = toloc
6051      elseif not inmath and item.id == 7 then % Apply recursively
6052        item.replace = item.replace and Babel.locale_map(item.replace)
6053        item.pre     = item.pre and Babel.locale_map(item.pre)
6054        item.post    = item.post and Babel.locale_map(item.post)
6055      elseif item.id == node.id'math' then
6056        inmath = (item.subtype == 0)
6057      end
6058    end
6059    return head
6060  end
6061 }
```

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

```
6062 \newcommand\babelcharproperty[1]{%
6063   \count@=#1\relax
6064   \ifvmode
6065     \expandafter\bbl@chprop
6066   \else
6067     \bbl@error{charproperty-only-vertical}{}{}{}%
6068   \fi}
6069 \newcommand\bbl@chprop[3][\the\count@]{%
6070   \@tempcnta=#1\relax
```

```
6071 \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6072   {\bbl@error{unknown-char-property}{}{#2}{}}%
6073   {}%
6074 \loop
6075   \bbl@cs{chprop@#2}{#3}%
6076 \ifnum\count@<\@tempcnta
6077   \advance\count@\@ne
6078 \repeat}
6079 \def\bbl@chprop@direction#1{%
6080   \directlua{
6081     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6082     Babel.characters[\the\count@]['d'] = '#1'
6083   }}
6084 \let\bbl@chprop@bc\bbl@chprop@direction
6085 \def\bbl@chprop@mirror#1{%
6086   \directlua{
6087     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6088     Babel.characters[\the\count@]['m'] = '\number#1'
6089   }}
6090 \let\bbl@chprop@bmg\bbl@chprop@mirror
6091 \def\bbl@chprop@linebreak#1{%
6092   \directlua{
6093     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6094     Babel.cjk_characters[\the\count@]['c'] = '#1'
6095   }}
6096 \let\bbl@chprop@lb\bbl@chprop@linebreak
6097 \def\bbl@chprop@locale#1{%
6098   \directlua{
6099     Babel.chr_to_loc = Babel.chr_to_loc or {}
6100     Babel.chr_to_loc[\the\count@] =
6101       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
6102   }}
```

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.

```
6103 \directlua{% DL7
6104   Babel.nohyphenation = \the\l@nohyphenation
6105 }
```

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

```
6106 \begingroup
6107 \catcode`\~=12
6108 \catcode`\%=12
6109 \catcode`\&=14
6110 \catcode`\|=12
6111 \gdef\babelprehyphenation{&%
6112   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
6113 \gdef\babelposthyphenation{&%
6114   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
6115 \gdef\bbl@settransform#1[#2]#3#4#5{&%
6116   \ifcase#1
6117     \bbl@activateprehyphen
6118   \or
6119     \bbl@activateposthyphen
6120   \fi
6121   \begingroup
```

127

```
6122        \def\babeltempa{\bbl@add@list\babeltempb}&%
6123        \let\babeltempb\@empty
6124        \def\bbl@tempa{#5}&%
6125        \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
6126        \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6127          \bbl@ifsamestring{##1}{remove}&%
6128            {\bbl@add@list\babeltempb{nil}}&%
6129            {\directlua{
6130               local rep = [=[##1]=]
6131               local three_args = '%s*=%s*([%-%d%.%a{}|]+)%s+([%-%d%.%a{}|]+)%s+([%-%d%.%a{}|]+)'
6132               &% Numeric passes directly: kern, penalty...
6133               rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6134               rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
6135               rep = rep:gsub('^%s*(after)%s*,', 'after = true, ')
6136               rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
6137               rep = rep:gsub('node%s*=%s*(%a+)%s*(%a*)', Babel.capture_node)
6138               rep = rep:gsub( '(norule)' .. three_args,
6139                   'norule = {' .. '%2, %3, %4' .. '}')
6140               if #1 == 0 or #1 == 2 then
6141                 rep = rep:gsub( '(space)' .. three_args,
6142                   'space = {' .. '%2, %3, %4' .. '}')
6143                 rep = rep:gsub( '(spacefactor)' .. three_args,
6144                   'spacefactor = {' .. '%2, %3, %4' .. '}')
6145                 rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
6146                 &% Transform values
6147                 rep, n = rep:gsub( '{([%a%-%.]+)|([%-%d%.]+)}',
6148                   '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6149               end
6150               if #1 == 1 then
6151                 rep = rep:gsub(   '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
6152                 rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
6153                 rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
6154               end
6155               tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
6156            }}}&%
6157        \bbl@foreach\babeltempb{&%
6158          \bbl@forkv{{##1}}{&%
6159            \in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6160              post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6161            \ifin@\else
6162              \bbl@error{bad-transform-option}{####1}{}{}&%
6163            \fi}}&%
6164        \let\bbl@kv@attribute\relax
6165        \let\bbl@kv@label\relax
6166        \let\bbl@kv@fonts\@empty
6167        \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6168        \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6169        \ifx\bbl@kv@attribute\relax
6170          \ifx\bbl@kv@label\relax\else
6171            \bbl@exp{\\\bbl@trim@def\\\bbl@kv@fonts{\bbl@kv@fonts}}&%
6172            \bbl@replace\bbl@kv@fonts{ }{,}&%
6173            \edef\bbl@kv@attribute{bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6174            \count@\z@
6175            \def\bbl@elt##1##2##3{&%
6176              \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6177                {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6178                    {\count@\@ne}&%
6179                    {\bbl@error{font-conflict-transforms}{}{}{}}}&%
6180                {}}&%
6181            \bbl@transfont@list
6182            \ifnum\count@=\z@
6183              \bbl@exp{\global\\\bbl@add\\\bbl@transfont@list
6184                {\\\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
```

```
6185            \fi
6186          \bbl@ifunset{\bbl@kv@attribute}&%
6187            {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6188            {}&%
6189          \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6190        \fi
6191      \else
6192        \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6193      \fi
6194      \directlua{
6195        local lbkr = Babel.linebreaking.replacements[#1]
6196        local u = unicode.utf8
6197        local id, attr, label
6198        if #1 == 0 then
6199          id = \the\csname bbl@id@@#3\endcsname\space
6200        else
6201          id = \the\csname l@#3\endcsname\space
6202        end
6203        \ifx\bbl@kv@attribute\relax
6204          attr = -1
6205        \else
6206          attr = luatexbase.registernumber'\bbl@kv@attribute'
6207        \fi
6208        \ifx\bbl@kv@label\relax\else  &% Same refs:
6209          label = [==[\bbl@kv@label]==]
6210        \fi
6211        &% Convert pattern:
6212        local patt = string.gsub([==[#4]==], '%s', '')
6213        if #1 == 0 then
6214          patt = string.gsub(patt, '|', ' ')
6215        end
6216        if not u.find(patt, '()', nil, true) then
6217          patt = '()' .. patt .. '()'
6218        end
6219        if #1 == 1 then
6220          patt = string.gsub(patt, '%(%)%^', '^()')
6221          patt = string.gsub(patt, '%$%(%)', '()$')
6222        end
6223        patt = u.gsub(patt, '{(.)}',
6224                function (n)
6225                  return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6226                end)
6227        patt = u.gsub(patt, '{(%x%x%x%x+)}',
6228                function (n)
6229                  return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6230                end)
6231        lbkr[id] = lbkr[id] or {}
6232        table.insert(lbkr[id],
6233          { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6234      }&%
6235    \endgroup}
6236 \endgroup
6237 \let\bbl@transfont@list\@empty
6238 \def\bbl@settransfont{%
6239    \global\let\bbl@settransfont\relax % Execute only once
6240    \gdef\bbl@transfont{%
6241      \def\bbl@elt####1####2####3{%
6242        \bbl@ifblank{####3}%
6243          {\count@\tw@}% Do nothing if no fonts
6244          {\count@\z@
6245           \bbl@vforeach{####3}{%
6246             \def\bbl@tempd{########1}%
6247             \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
```

```
6248            \ifx\bbl@tempd\bbl@tempe
6249              \count@\@ne
6250            \else\ifx\bbl@tempd\bbl@transfam
6251              \count@\@ne
6252            \fi\fi}%
6253          \ifcase\count@
6254            \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6255          \or
6256            \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6257          \fi}}%
6258        \bbl@transfont@list}%
6259    \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6260    \gdef\bbl@transfam{-unknown-}%
6261    \bbl@foreach\bbl@font@fams{%
6262      \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6263      \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6264        {\xdef\bbl@transfam{##1}}%
6265        {}}}
6266  \DeclareRobustCommand\enablelocaletransform[1]{%
6267    \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6268      {\bbl@error{transform-not-available}{#1}{}{}}%
6269      {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6270  \DeclareRobustCommand\disablelocaletransform[1]{%
6271    \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6272      {\bbl@error{transform-not-available-b}{#1}{}{}}%
6273      {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6274  \def\bbl@activateposthyphen{%
6275    \let\bbl@activateposthyphen\relax
6276    \directlua{
6277      require('babel-transforms.lua')
6278      Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6279    }}
6280  \def\bbl@activateprehyphen{%
6281    \let\bbl@activateprehyphen\relax
6282    \directlua{
6283      require('babel-transforms.lua')
6284      Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6285    }}
6286  \newcommand\SetTransformValue[3]{%
6287    \directlua{
6288      Babel.locale_props[\the\csname bbl@id@@#1\endcsname].vars["#2"] = #3
6289    }}
```

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.

```
6290  \newcommand\localeprehyphenation[1]{%
6291    \directlua{ Babel.string_prehyphenation([==[#1]==], \the\localeid) }}
```

10.11. Bidi

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

```
6292  \def\bbl@activate@preotf{%
6293    \let\bbl@activate@preotf\relax  % only once
6294    \directlua{
6295      function Babel.pre_otfload_v(head)
6296        if Babel.numbers and Babel.digits_mapped then
6297          head = Babel.numbers(head)
6298        end
```

130

```
6299        if Babel.bidi_enabled then
6300           head = Babel.bidi(head, false, dir)
6301        end
6302        return head
6303      end
6304      %
6305      function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6306        if Babel.numbers and Babel.digits_mapped then
6307           head = Babel.numbers(head)
6308        end
6309        if Babel.bidi_enabled then
6310           head = Babel.bidi(head, false, dir)
6311        end
6312        return head
6313      end
6314      %
6315      luatexbase.add_to_callback('pre_linebreak_filter',
6316        Babel.pre_otfload_v,
6317        'Babel.pre_otfload_v',
6318        luatexbase.priority_in_callback('pre_linebreak_filter',
6319           'luaotfload.node_processor') or nil)
6320      %
6321      luatexbase.add_to_callback('hpack_filter',
6322        Babel.pre_otfload_h,
6323        'Babel.pre_otfload_h',
6324        luatexbase.priority_in_callback('hpack_filter',
6325           'luaotfload.node_processor') or nil)
6326 }}
```

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

```
6327 \breakafterdirmode=1
6328 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6329   \let\bbl@beforeforeign\leavevmode
6330   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6331   \RequirePackage{luatexbase}
6332   \bbl@activate@preotf
6333   \directlua{
6334     require('babel-data-bidi.lua')
6335     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6336       require('babel-bidi-basic.lua')
6337     \or
6338       require('babel-bidi-basic-r.lua')
6339       table.insert(Babel.ranges, {0xE000,   0xF8FF, 'on'})
6340       table.insert(Babel.ranges, {0xF0000,  0xFFFFD, 'on'})
6341       table.insert(Babel.ranges, {0x100000, 0x10FFFD, 'on'})
6342     \fi}
6343   \newattribute\bbl@attr@dir
6344   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6345   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6346 \fi
6347 \chardef\bbl@thetextdir\z@
6348 \chardef\bbl@thepardir\z@
6349 \def\bbl@getluadir#1{%
6350   \directlua{
6351     if tex.#1dir == 'TLT' then
6352       tex.sprint('0')
6353     elseif tex.#1dir == 'TRT' then
6354       tex.sprint('1')
6355     end}}
6356 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
```

```
6357  \ifcase#3\relax
6358    \ifcase\bbl@getluadir{#1}\relax\else
6359      #2 TLT\relax
6360    \fi
6361  \else
6362    \ifcase\bbl@getluadir{#1}\relax
6363      #2 TRT\relax
6364    \fi
6365  \fi}
6366 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6367 \def\bbl@thedir{0}
6368 \def\bbl@textdir#1{%
6369   \bbl@setluadir{text}\textdir{#1}%
6370   \chardef\bbl@thetextdir#1\relax
6371   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6372   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6373 \def\bbl@pardir#1{% Used twice
6374   \bbl@setluadir{par}\pardir{#1}%
6375   \chardef\bbl@thepardir#1\relax}
6376 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}%   Used once
6377 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}%   Unused
6378 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once
```

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

```
6379 \ifnum\bbl@bidimode>\z@ % Any bidi=
6380   \def\bbl@insidemath{0}%
6381   \def\bbl@everymath{\def\bbl@insidemath{1}}
6382   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6383   \frozen@everymath\expandafter{%
6384     \expandafter\bbl@everymath\the\frozen@everymath}
6385   \frozen@everydisplay\expandafter{%
6386     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6387   \AtBeginDocument{
6388     \directlua{
6389       function Babel.math_box_dir(head)
6390         if not (token.get_macro('bbl@insidemath') == '0') then
6391           if Babel.hlist_has_bidi(head) then
6392             local d = node.new(node.id'dir')
6393             d.dir = '+TRT'
6394             node.insert_before(head, node.has_glyph(head), d)
6395             local inmath = false
6396             for item in node.traverse(head) do
6397               if item.id == 11 then
6398                 inmath = (item.subtype == 0)
6399               elseif not inmath then
6400                 node.set_attribute(item,
6401                   Babel.attr_dir, token.get_macro('bbl@thedir'))
6402               end
6403             end
6404           end
6405         end
6406         return head
6407       end
6408       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6409         "Babel.math_box_dir", 0)
6410       if Babel.unset_atdir then
6411         luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6412           "Babel.unset_atdir")
6413         luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6414           "Babel.unset_atdir")
6415       end
6416 }}%
```

132

```
6417 \fi
```

Experimental. Tentative name.

```
6418 \DeclareRobustCommand\localebox[1]{%
6419   {\def\bbl@insidemath{0}%
6420     \mbox{\foreignlanguage{\languagename}{#1}}}}
```

10.12.Layout

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

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

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

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

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

```
6421 \bbl@trace{Redefinitions for bidi layout}
6422 %
6423 ⟨⟨*More package options⟩⟩ ≡
6424 \chardef\bbl@eqnpos\z@
6425 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6426 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6427 ⟨⟨/More package options⟩⟩
6428 %
6429 \ifnum\bbl@bidimode>\z@ % Any bidi=
6430   \matheqdirmode\@ne % A luatex primitive
6431   \let\bbl@eqnodir\relax
6432   \def\bbl@eqdel{()}
6433   \def\bbl@eqnum{%
6434     {\normalfont\normalcolor
6435      \expandafter\@firstoftwo\bbl@eqdel
6436      \theequation
6437      \expandafter\@secondoftwo\bbl@eqdel}}
6438   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6439   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6440   \def\bbl@eqno@flip#1{%
6441     \ifdim\predisplaysize=-\maxdimen
6442       \eqno
6443       \hb@xt@.01pt{%
6444         \hb@xt@\displaywidth{\hss{#1\glet\bbl@upset\@currentlabel}}\hss}%
6445     \else
6446       \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6447     \fi
6448     \bbl@exp{\def\\\@currentlabel{\[bbl@upset]}}}
6449   \def\bbl@leqno@flip#1{%
6450     \ifdim\predisplaysize=-\maxdimen
6451       \leqno
6452       \hb@xt@.01pt{%
6453         \hss\hb@xt@\displaywidth{{#1\glet\bbl@upset\@currentlabel}\hss}}%
6454     \else
6455       \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
```

```
6456        \fi
6457        \bbl@exp{\def\\\@currentlabel{\[bbl@upset]}}}
6458    \AtBeginDocument{%
6459      \ifx\bbl@noamsmath\relax\else
6460      \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6461        \AddToHook{env/equation/begin}{%
6462          \ifnum\bbl@thetextdir>\z@
6463            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6464            \let\@eqnnum\bbl@eqnum
6465            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6466            \chardef\bbl@thetextdir\z@
6467            \bbl@add\normalfont{\bbl@eqnodir}%
6468            \ifcase\bbl@eqnpos
6469              \let\bbl@puteqno\bbl@eqno@flip
6470            \or
6471              \let\bbl@puteqno\bbl@leqno@flip
6472            \fi
6473          \fi}%
6474        \ifnum\bbl@eqnpos=\tw@\else
6475          \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6476        \fi
6477        \AddToHook{env/eqnarray/begin}{%
6478          \ifnum\bbl@thetextdir>\z@
6479            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6480            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6481            \chardef\bbl@thetextdir\z@
6482            \bbl@add\normalfont{\bbl@eqnodir}%
6483            \ifnum\bbl@eqnpos=\@ne
6484              \def\@eqnnum{%
6485                \setbox\z@\hbox{\bbl@eqnum}%
6486                \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6487            \else
6488              \let\@eqnnum\bbl@eqnum
6489            \fi
6490          \fi}
6491        % Hack. YA luatex bug?:
6492        \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6493      \else % amstex
6494        \bbl@exp{% Hack to hide maybe undefined conditionals:
6495          \chardef\bbl@eqnpos=0%
6496            \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6497        \ifnum\bbl@eqnpos=\@ne
6498          \let\bbl@ams@lap\hbox
6499        \else
6500          \let\bbl@ams@lap\llap
6501        \fi
6502        \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6503        \bbl@sreplace\intertext@{\normalbaselines}%
6504          {\normalbaselines
6505            \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6506        \ExplSyntaxOff
6507        \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6508        \ifx\bbl@ams@lap\hbox % leqno
6509          \def\bbl@ams@flip#1{%
6510            \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6511        \else % eqno
6512          \def\bbl@ams@flip#1{%
6513            \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6514        \fi
6515        \def\bbl@ams@preset#1{%
6516          \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6517          \ifnum\bbl@thetextdir>\z@
6518            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
```

```
6519        \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6520        \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6521      \fi}%
6522    \ifnum\bbl@eqnpos=\tw@\else
6523      \def\bbl@ams@equation{%
6524        \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6525        \ifnum\bbl@thetextdir>\z@
6526          \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6527          \chardef\bbl@thetextdir\z@
6528          \bbl@add\normalfont{\bbl@eqnodir}%
6529          \ifcase\bbl@eqnpos
6530            \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6531          \or
6532            \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6533          \fi
6534        \fi}%
6535      \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6536      \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6537    \fi
6538    \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6539    \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6540    \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6541    \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6542    \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6543    \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6544    \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6545    \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6546    \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6547    % Hackish, for proper alignment. Don't ask me why it works!:
6548    \bbl@exp{% Avoid a 'visible' conditional
6549      \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}%
6550      \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6551    \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6552    \AddToHook{env/split/before}{%
6553      \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6554      \ifnum\bbl@thetextdir>\z@
6555        \bbl@ifsamestring\@currenvir{equation}%
6556          {\ifx\bbl@ams@lap\hbox % leqno
6557            \def\bbl@ams@flip#1{%
6558              \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6559          \else
6560            \def\bbl@ams@flip#1{%
6561              \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6562          \fi}%
6563          {}%
6564      \fi}%
6565    \fi\fi}
6566 \fi
6567 \def\bbl@provide@extra#1{%
6568    % == onchar ==
6569    \ifx\bbl@KVP@onchar\@nnil\else
6570      \bbl@luahyphenate
6571      \bbl@exp{%
6572        \\\AddToHook{env/document/before}{{\\\select@language{#1}{}}}}%
6573      \directlua{
6574        if Babel.locale_mapped == nil then
6575          Babel.locale_mapped = true
6576          Babel.linebreaking.add_before(Babel.locale_map, 1)
6577          Babel.loc_to_scr = {}
6578          Babel.chr_to_loc = Babel.chr_to_loc or {}
6579        end
6580        Babel.locale_props[\the\localeid].letters = false
6581      }%
```

```
6582    \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6583    \ifin@
6584      \directlua{
6585        Babel.locale_props[\the\localeid].letters = true
6586      }%
6587    \fi
6588    \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6589    \ifin@
6590      \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6591        \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
6592      \fi
6593      \bbl@exp{\\\bbl@add\\\bbl@starthyphens
6594        {\\\bbl@patterns@lua{\languagename}}}%
6595      %^^A add error/warning if no script
6596      \directlua{
6597        if Babel.script_blocks['\bbl@cl{sbcp}'] then
6598          Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcp}']
6599          Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
6600        end
6601      }%
6602    \fi
6603    \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6604    \ifin@
6605      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
6606      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
6607      \directlua{
6608        if Babel.script_blocks['\bbl@cl{sbcp}'] then
6609          Babel.loc_to_scr[\the\localeid] =
6610            Babel.script_blocks['\bbl@cl{sbcp}']
6611        end}%
6612      \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
6613        \AtBeginDocument{%
6614          \bbl@patchfont{{\bbl@mapselect}}%
6615          {\selectfont}}%
6616        \def\bbl@mapselect{%
6617          \let\bbl@mapselect\relax
6618          \edef\bbl@prefontid{\fontid\font}}%
6619        \def\bbl@mapdir##1{%
6620          \begingroup
6621            \setbox\z@\hbox{% Force text mode
6622              \def\languagename{##1}%
6623              \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6624              \bbl@switchfont
6625              \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6626                \directlua{
6627                  Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
6628                    ['/\bbl@prefontid'] = \fontid\font\space}%
6629              \fi}%
6630          \endgroup}%
6631      \fi
6632      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
6633    \fi
6634    % TODO - catch non-valid values
6635  \fi
6636  % == mapfont ==
6637  % For bidi texts, to switch the font based on direction
6638  \ifx\bbl@KVP@mapfont\@nnil\else
6639    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
6640      {\bbl@error{unknown-mapfont}{}{}{}}%
6641    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
6642    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
6643    \ifx\bbl@mapselect\@undefined % TODO. See onchar.
6644      \AtBeginDocument{%
```

```
6645        \bbl@patchfont{{\bbl@mapselect}}%
6646          {\selectfont}}%
6647      \def\bbl@mapselect{%
6648        \let\bbl@mapselect\relax
6649        \edef\bbl@prefontid{\fontid\font}}%
6650      \def\bbl@mapdir##1{%
6651        {\def\languagename{##1}%
6652         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6653         \bbl@switchfont
6654         \directlua{Babel.fontmap
6655           [\the\csname bbl@wdir@##1\endcsname]%
6656           [\bbl@prefontid]=\fontid\font}}}%
6657    \fi
6658    \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
6659  \fi
6660  % == Line breaking: CJK quotes == %^^A -> @extras
6661  \ifcase\bbl@engine\or
6662    \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
6663    \ifin@
6664      \bbl@ifunset{bbl@quote@\languagename}{}%
6665        {\directlua{
6666          Babel.locale_props[\the\localeid].cjk_quotes = {}
6667          local cs = 'op'
6668          for c in string.utfvalues(%
6669              [[\csname bbl@quote@\languagename\endcsname]]) do
6670            if Babel.cjk_characters[c].c == 'qu' then
6671              Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6672            end
6673            cs = ( cs == 'op') and 'cl' or 'op'
6674          end
6675        }}%
6676    \fi
6677  \fi
6678  % == Counters: mapdigits ==
6679  % Native digits
6680  \ifx\bbl@KVP@mapdigits\@nnil\else
6681    \bbl@ifunset{bbl@dgnat@\languagename}{}%
6682      {\RequirePackage{luatexbase}%
6683       \bbl@activate@preotf
6684       \directlua{
6685         Babel.digits_mapped = true
6686         Babel.digits = Babel.digits or {}
6687         Babel.digits[\the\localeid] =
6688           table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6689         if not Babel.numbers then
6690           function Babel.numbers(head)
6691             local LOCALE = Babel.attr_locale
6692             local GLYPH = node.id'glyph'
6693             local inmath = false
6694             for item in node.traverse(head) do
6695               if not inmath and item.id == GLYPH then
6696                 local temp = node.get_attribute(item, LOCALE)
6697                 if Babel.digits[temp] then
6698                   local chr = item.char
6699                   if chr > 47 and chr < 58 then
6700                     item.char = Babel.digits[temp][chr-47]
6701                   end
6702                 end
6703               elseif item.id == node.id'math' then
6704                 inmath = (item.subtype == 0)
6705               end
6706             end
6707             return head
```

```
6708          end
6709        end
6710      }}%
6711  \fi
6712  % == transforms ==
6713  \ifx\bbl@KVP@transforms\@nnil\else
6714    \def\bbl@elt##1##2##3{%
6715      \in@{$transforms.}{$##1}%
6716      \ifin@
6717        \def\bbl@tempa{##1}%
6718        \bbl@replace\bbl@tempa{transforms.}{}%
6719        \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6720      \fi}%
6721    \bbl@exp{%
6722      \\\bbl@ifblank{\bbl@cl{dgnat}}%
6723       {\let\\\bbl@tempa\relax}%
6724       {\def\\\bbl@tempa{%
6725         \\\bbl@elt{transforms.prehyphenation}%
6726          {digits.native.1.0}{([0-9])}%
6727         \\\bbl@elt{transforms.prehyphenation}%
6728          {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}}}%
6729    \ifx\bbl@tempa\relax\else
6730      \toks@\expandafter\expandafter\expandafter{%
6731        \csname bbl@inidata@\languagename\endcsname}%
6732      \bbl@csarg\edef{inidata@\languagename}{%
6733        \unexpanded\expandafter{\bbl@tempa}%
6734        \the\toks@}%
6735    \fi
6736    \csname bbl@inidata@\languagename\endcsname
6737    \bbl@release@transforms\relax % \relax closes the last item.
6738  \fi}
```

Start tabular here:

```
6739 \def\localerestoredirs{%
6740   \ifcase\bbl@thetextdir
6741     \ifnum\textdirection=\z@\else\textdir TLT\fi
6742   \else
6743     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6744   \fi
6745   \ifcase\bbl@thepardir
6746     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6747   \else
6748     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6749   \fi}
6750 \IfBabelLayout{tabular}%
6751   {\chardef\bbl@tabular@mode\tw@}% All RTL
6752   {\IfBabelLayout{notabular}%
6753     {\chardef\bbl@tabular@mode\z@}%
6754     {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6755 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6756   % Redefine: vrules mess up dirs. TODO: why?
6757   \def\@arstrut{\relax\copy\@arstrutbox}%
6758   \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6759     \let\bbl@parabefore\relax
6760     \AddToHook{para/before}{\bbl@parabefore}
6761     \AtBeginDocument{%
6762       \bbl@replace\@tabular{$}{$%
6763         \def\bbl@insidemath{0}%
6764         \def\bbl@parabefore{\localerestoredirs}}%
6765     \ifnum\bbl@tabular@mode=\@ne
6766       \bbl@ifunset{@tabclassz}{}{%
6767         \bbl@exp{% Hide conditionals
6768           \\\bbl@sreplace\\\@tabclassz
```

138

```
6769            {\<ifcase>\\\@chnum}%
6770            {\\\localerestoredirs\<ifcase>\\\@chnum}}%
6771        \@ifpackageloaded{colortbl}%
6772          {\bbl@sreplace\@classz
6773            {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6774          {\@ifpackageloaded{array}%
6775            {\bbl@exp{% Hide conditionals
6776                \\\bbl@sreplace\\\@classz
6777                  {\<ifcase>\\\@chnum}%
6778                  {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6779                \\\bbl@sreplace\\\@classz
6780                  {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
6781            {}}%
6782      \fi}%
6783  \or % 2 = All RTL - tabular
6784      \let\bbl@parabefore\relax
6785      \AddToHook{para/before}{\bbl@parabefore}%
6786      \AtBeginDocument{%
6787        \@ifpackageloaded{colortbl}%
6788          {\bbl@replace\@tabular{$}{$%
6789            \def\bbl@insidemath{0}%
6790            \def\bbl@parabefore{\localerestoredirs}}%
6791          \bbl@sreplace\@classz
6792            {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6793          {}}%
6794  \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.

6795  \AtBeginDocument{%
6796      \@ifpackageloaded{multicol}%
6797        {\toks@\expandafter{\multi@column@out}%
6798         \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6799        {}%
6800      \@ifpackageloaded{paracol}%
6801        {\edef\pcol@output{%
6802         \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6803        {}}%
6804 \fi
6805 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout

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

6806 \ifnum\bbl@bidimode>\z@ % Any bidi=
6807  \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6808    \bbl@exp{%
6809      \mathdir\the\bodydir
6810      #1%             Once entered in math, set boxes to restore values
6811      \def\\\bbl@insidemath{0}%
6812      \<ifmmode>%
6813        \everyvbox{%
6814          \the\everyvbox
6815          \bodydir\the\bodydir
6816          \mathdir\the\mathdir
6817          \everyhbox{\the\everyhbox}%
6818          \everyvbox{\the\everyvbox}}%
6819        \everyhbox{%
6820          \the\everyhbox
6821          \bodydir\the\bodydir
6822          \mathdir\the\mathdir
6823          \everyhbox{\the\everyhbox}%
```

```
6824        \everyvbox{\the\everyvbox}}%
6825      \<fi>}}%
6826    \def\@hangfrom#1{%
6827      \setbox\@tempboxa\hbox{{#1}}%
6828      \hangindent\wd\@tempboxa
6829      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6830        \shapemode\@ne
6831      \fi
6832      \noindent\box\@tempboxa}
6833  \fi
6834  \IfBabelLayout{tabular}
6835    {\let\bbl@OL@@tabular\@tabular
6836     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6837     \let\bbl@NL@@tabular\@tabular
6838     \AtBeginDocument{%
6839       \ifx\bbl@NL@@tabular\@tabular\else
6840         \bbl@exp{\\\in@{\\\bbl@nextfake}{\[@tabular]}}%
6841         \ifin@\else
6842           \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6843         \fi
6844         \let\bbl@NL@@tabular\@tabular
6845       \fi}}
6846      {}
6847  \IfBabelLayout{lists}
6848    {\let\bbl@OL@list\list
6849     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6850     \let\bbl@NL@list\list
6851     \def\bbl@listparshape#1#2#3{%
6852       \parshape #1 #2 #3 %
6853       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6854         \shapemode\tw@
6855       \fi}}
6856      {}
6857  \IfBabelLayout{graphics}
6858    {\let\bbl@pictresetdir\relax
6859     \def\bbl@pictsetdir#1{%
6860       \ifcase\bbl@thetextdir
6861         \let\bbl@pictresetdir\relax
6862       \else
6863         \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6864           \or\textdir TLT
6865           \else\bodydir TLT \textdir TLT
6866         \fi
6867         % \(text|par)dir required in pgf:
6868         \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6869       \fi}%
6870     \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6871     \directlua{
6872       Babel.get_picture_dir = true
6873       Babel.picture_has_bidi = 0
6874       %
6875       function Babel.picture_dir (head)
6876         if not Babel.get_picture_dir then return head end
6877         if Babel.hlist_has_bidi(head) then
6878           Babel.picture_has_bidi = 1
6879         end
6880         return head
6881       end
6882       luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6883         "Babel.picture_dir")
6884     }%
6885     \AtBeginDocument{%
6886       \def\LS@rot{%
```

```
6887        \setbox\@outputbox\vbox{%
6888          \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}%
6889      \long\def\put(#1,#2)#3{%
6890        \@killglue
6891        % Try:
6892        \ifx\bbl@pictresetdir\relax
6893          \def\bbl@tempc{0}%
6894        \else
6895          \directlua{
6896            Babel.get_picture_dir = true
6897            Babel.picture_has_bidi = 0
6898          }%
6899          \setbox\z@\hb@xt@\z@{%
6900            \@defaultunitsset\@tempdimc{#1}\unitlength
6901            \kern\@tempdimc
6902            #3\hss}% TODO: #3 executed twice (below). That's bad.
6903          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6904        \fi
6905        % Do:
6906        \@defaultunitsset\@tempdimc{#2}\unitlength
6907        \raise\@tempdimc\hb@xt@\z@{%
6908          \@defaultunitsset\@tempdimc{#1}\unitlength
6909          \kern\@tempdimc
6910          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6911        \ignorespaces}%
6912      \MakeRobust\put}%
6913    \AtBeginDocument
6914      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6915        \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6916          \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6917          \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6918          \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6919        \fi
6920        \ifx\tikzpicture\@undefined\else
6921          \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6922          \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6923          \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6924        \fi
6925        \ifx\tcolorbox\@undefined\else
6926          \def\tcb@drawing@env@begin{%
6927            \csname tcb@before@\tcb@split@state\endcsname
6928            \bbl@pictsetdir\tw@
6929            \begin{\kvtcb@graphenv}%
6930            \tcb@bbdraw
6931            \tcb@apply@graph@patches}%
6932          \def\tcb@drawing@env@end{%
6933            \end{\kvtcb@graphenv}%
6934            \bbl@pictresetdir
6935            \csname tcb@after@\tcb@split@state\endcsname}%
6936        \fi
6937      }}
6938    {}
```

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.

```
6939 \IfBabelLayout{counters*}%
6940   {\bbl@add\bbl@opt@layout{.counters.}%
6941    \directlua{
6942      luatexbase.add_to_callback("process_output_buffer",
6943        Babel.discard_sublr , "Babel.discard_sublr") }%
6944   }{}
6945 \IfBabelLayout{counters}%
```

```
6946    {\let\bbl@OL@@textsuperscript\@textsuperscript
6947    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6948    \let\bbl@latinarabic=\@arabic
6949    \let\bbl@OL@@arabic\@arabic
6950    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6951    \@ifpackagewith{babel}{bidi=default}%
6952      {\let\bbl@asciiroman=\@roman
6953      \let\bbl@OL@@roman\@roman
6954      \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6955      \let\bbl@asciiRoman=\@Roman
6956      \let\bbl@OL@@roman\@Roman
6957      \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6958      \let\bbl@OL@labelenumii\labelenumii
6959      \def\labelenumii{)\theenumii(}%
6960      \let\bbl@OL@p@enumiii\p@enumiii
6961      \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6962 <@Footnote changes@>
6963 \IfBabelLayout{footnotes}%
6964    {\let\bbl@OL@footnote\footnote
6965    \BabelFootnote\footnote\languagename{}{}%
6966    \BabelFootnote\localfootnote\languagename{}{}%
6967    \BabelFootnote\mainfootnote{}{}{}}
6968    {}
```

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

```
6969 \IfBabelLayout{extras}%
6970    {\bbl@ncarg\let\bbl@OL@underline{underline }%
6971    \bbl@carg\bbl@sreplace{underline }%
6972      {$\@@underline}{\bgroup\bbl@nextfake$\@@underline}%
6973    \bbl@carg\bbl@sreplace{underline }%
6974      {\m@th$}{\m@th$\egroup}%
6975    \let\bbl@OL@LaTeXe\LaTeXe
6976    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6977      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6978      \babelsublr{%
6979        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6980    {}
6981 ⟨/luatex⟩
```

10.13. Lua: transforms

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

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

```
6982 ⟨*transforms⟩
6983 Babel.linebreaking.replacements = {}
6984 Babel.linebreaking.replacements[0] = {}  -- pre
6985 Babel.linebreaking.replacements[1] = {}  -- post
6986
6987 function Babel.tovalue(v)
6988   if type(v) == 'table' then
6989     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
6990   else
6991     return v
```

```lua
6992    end
6993 end
6994
6995 Babel.fetch_subtext = {}
6996
6997 Babel.ignore_pre_char = function(node)
6998    return (node.lang == Babel.nohyphenation)
6999 end
7000
7001 -- Merging both functions doesn't seen feasible, because there are too
7002 -- many differences.
7003 Babel.fetch_subtext[0] = function(head)
7004    local word_string = ''
7005    local word_nodes = {}
7006    local lang
7007    local item = head
7008    local inmath = false
7009
7010    while item do
7011
7012      if item.id == 11 then
7013        inmath = (item.subtype == 0)
7014      end
7015
7016      if inmath then
7017        -- pass
7018
7019      elseif item.id == 29 then
7020        local locale = node.get_attribute(item, Babel.attr_locale)
7021
7022        if lang == locale or lang == nil then
7023          lang = lang or locale
7024          if Babel.ignore_pre_char(item) then
7025            word_string = word_string .. Babel.us_char
7026          else
7027            word_string = word_string .. unicode.utf8.char(item.char)
7028          end
7029          word_nodes[#word_nodes+1] = item
7030        else
7031          break
7032        end
7033
7034      elseif item.id == 12 and item.subtype == 13 then
7035        word_string = word_string .. ' '
7036        word_nodes[#word_nodes+1] = item
7037
7038      -- Ignore leading unrecognized nodes, too.
7039      elseif word_string ~= '' then
7040        word_string = word_string .. Babel.us_char
7041        word_nodes[#word_nodes+1] = item  -- Will be ignored
7042      end
7043
7044      item = item.next
7045    end
7046
7047    -- Here and above we remove some trailing chars but not the
7048    -- corresponding nodes. But they aren't accessed.
7049    if word_string:sub(-1) == ' ' then
7050      word_string = word_string:sub(1,-2)
7051    end
7052    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7053    return word_string, word_nodes, item, lang
7054 end
```

```
7055
7056 Babel.fetch_subtext[1] = function(head)
7057   local word_string = ''
7058   local word_nodes = {}
7059   local lang
7060   local item = head
7061   local inmath = false
7062
7063   while item do
7064
7065     if item.id == 11 then
7066       inmath = (item.subtype == 0)
7067     end
7068
7069     if inmath then
7070       -- pass
7071
7072     elseif item.id == 29 then
7073       if item.lang == lang or lang == nil then
7074         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7075           lang = lang or item.lang
7076           word_string = word_string .. unicode.utf8.char(item.char)
7077           word_nodes[#word_nodes+1] = item
7078         end
7079       else
7080         break
7081       end
7082
7083     elseif item.id == 7 and item.subtype == 2 then
7084       word_string = word_string .. '='
7085       word_nodes[#word_nodes+1] = item
7086
7087     elseif item.id == 7 and item.subtype == 3 then
7088       word_string = word_string .. '|'
7089       word_nodes[#word_nodes+1] = item
7090
7091     -- (1) Go to next word if nothing was found, and (2) implicitly
7092     -- remove leading USs.
7093     elseif word_string == '' then
7094       -- pass
7095
7096     -- This is the responsible for splitting by words.
7097     elseif (item.id == 12 and item.subtype == 13) then
7098       break
7099
7100     else
7101       word_string = word_string .. Babel.us_char
7102       word_nodes[#word_nodes+1] = item  -- Will be ignored
7103     end
7104
7105     item = item.next
7106   end
7107
7108   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7109   return word_string, word_nodes, item, lang
7110 end
7111
7112 function Babel.pre_hyphenate_replace(head)
7113   Babel.hyphenate_replace(head, 0)
7114 end
7115
7116 function Babel.post_hyphenate_replace(head)
7117   Babel.hyphenate_replace(head, 1)
```

```lua
7118 end
7119
7120 Babel.us_char = string.char(31)
7121
7122 function Babel.hyphenate_replace(head, mode)
7123   local u = unicode.utf8
7124   local lbkr = Babel.linebreaking.replacements[mode]
7125   local tovalue = Babel.tovalue
7126
7127   local word_head = head
7128
7129   while true do  -- for each subtext block
7130
7131     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7132
7133     if Babel.debug then
7134       print()
7135       print((mode == 0) and '@@@@<' or '@@@@>', w)
7136     end
7137
7138     if nw == nil and w == '' then break end
7139
7140     if not lang then goto next end
7141     if not lbkr[lang] then goto next end
7142
7143     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7144     -- loops are nested.
7145     for k=1, #lbkr[lang] do
7146       local p = lbkr[lang][k].pattern
7147       local r = lbkr[lang][k].replace
7148       local attr = lbkr[lang][k].attr or -1
7149
7150       if Babel.debug then
7151         print('*****', p, mode)
7152       end
7153
7154       -- This variable is set in some cases below to the first *byte*
7155       -- after the match, either as found by u.match (faster) or the
7156       -- computed position based on sc if w has changed.
7157       local last_match = 0
7158       local step = 0
7159
7160       -- For every match.
7161       while true do
7162         if Babel.debug then
7163           print('=====')
7164         end
7165         local new  -- used when inserting and removing nodes
7166         local dummy_node -- used by after
7167
7168         local matches = { u.match(w, p, last_match) }
7169
7170         if #matches < 2 then break end
7171
7172         -- Get and remove empty captures (with ()'s, which return a
7173         -- number with the position), and keep actual captures
7174         -- (from (...)), if any, in matches.
7175         local first = table.remove(matches, 1)
7176         local last  = table.remove(matches, #matches)
7177         -- Non re-fetched substrings may contain \31, which separates
7178         -- subsubstrings.
7179         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7180
```

```
7181          local save_last = last -- with A()BC()D, points to D
7182
7183          -- Fix offsets, from bytes to unicode. Explained above.
7184          first = u.len(w:sub(1, first-1)) + 1
7185          last  = u.len(w:sub(1, last-1)) -- now last points to C
7186
7187          -- This loop stores in a small table the nodes
7188          -- corresponding to the pattern. Used by 'data' to provide a
7189          -- predictable behavior with 'insert' (w_nodes is modified on
7190          -- the fly), and also access to 'remove'd nodes.
7191          local sc = first-1            -- Used below, too
7192          local data_nodes = {}
7193
7194          local enabled = true
7195          for q = 1, last-first+1 do
7196            data_nodes[q] = w_nodes[sc+q]
7197            if enabled
7198               and attr > -1
7199               and not node.has_attribute(data_nodes[q], attr)
7200            then
7201               enabled = false
7202            end
7203          end
7204
7205          -- This loop traverses the matched substring and takes the
7206          -- corresponding action stored in the replacement list.
7207          -- sc = the position in substr nodes / string
7208          -- rc = the replacement table index
7209          local rc = 0
7210
7211 ------- TODO. dummy_node?
7212          while rc < last-first+1 or dummy_node do -- for each replacement
7213            if Babel.debug then
7214              print('.....', rc + 1)
7215            end
7216            sc = sc + 1
7217            rc = rc + 1
7218
7219            if Babel.debug then
7220              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7221              local ss = ''
7222              for itt in node.traverse(head) do
7223               if itt.id == 29 then
7224                 ss = ss .. unicode.utf8.char(itt.char)
7225               else
7226                 ss = ss .. '{' .. itt.id .. '}'
7227               end
7228              end
7229              print('*****************', ss)
7230
7231            end
7232
7233            local crep = r[rc]
7234            local item = w_nodes[sc]
7235            local item_base = item
7236            local placeholder = Babel.us_char
7237            local d
7238
7239            if crep and crep.data then
7240              item_base = data_nodes[crep.data]
7241            end
7242
7243            if crep then
```

```
7244              step = crep.step or step
7245            end
7246
7247          if crep and crep.after then
7248            crep.insert = true
7249            if dummy_node then
7250              item = dummy_node
7251            else -- TODO. if there is a node after?
7252              d = node.copy(item_base)
7253              head, item = node.insert_after(head, item, d)
7254              dummy_node = item
7255            end
7256          end
7257
7258          if crep and not crep.after and dummy_node then
7259            node.remove(head, dummy_node)
7260            dummy_node = nil
7261          end
7262
7263          if (not enabled) or (crep and next(crep) == nil) then -- = {}
7264            if step == 0 then
7265              last_match = save_last     -- Optimization
7266            else
7267              last_match = utf8.offset(w, sc+step)
7268            end
7269            goto next
7270
7271          elseif crep == nil or crep.remove then
7272            node.remove(head, item)
7273            table.remove(w_nodes, sc)
7274            w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7275            sc = sc - 1  -- Nothing has been inserted.
7276            last_match = utf8.offset(w, sc+1+step)
7277            goto next
7278
7279          elseif crep and crep.kashida then -- Experimental
7280            node.set_attribute(item,
7281               Babel.attr_kashida,
7282               crep.kashida)
7283            last_match = utf8.offset(w, sc+1+step)
7284            goto next
7285
7286          elseif crep and crep.string then
7287            local str = crep.string(matches)
7288            if str == '' then  -- Gather with nil
7289              node.remove(head, item)
7290              table.remove(w_nodes, sc)
7291              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7292              sc = sc - 1  -- Nothing has been inserted.
7293            else
7294              local loop_first = true
7295              for s in string.utfvalues(str) do
7296                d = node.copy(item_base)
7297                d.char = s
7298                if loop_first then
7299                  loop_first = false
7300                  head, new = node.insert_before(head, item, d)
7301                  if sc == 1 then
7302                    word_head = head
7303                  end
7304                  w_nodes[sc] = d
7305                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7306                else
```

```
7307              sc = sc + 1
7308              head, new = node.insert_before(head, item, d)
7309              table.insert(w_nodes, sc, new)
7310              w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7311            end
7312            if Babel.debug then
7313              print('.....', 'str')
7314              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7315            end
7316          end  -- for
7317          node.remove(head, item)
7318        end  -- if ''
7319        last_match = utf8.offset(w, sc+1+step)
7320        goto next
7321
7322      elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7323        d = node.new(7, 3)   -- (disc, regular)
7324        d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
7325        d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
7326        d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7327        d.attr = item_base.attr
7328        if crep.pre == nil then  -- TeXbook p96
7329          d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7330        else
7331          d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7332        end
7333        placeholder = '|'
7334        head, new = node.insert_before(head, item, d)
7335
7336      elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7337        -- ERROR
7338
7339      elseif crep and crep.penalty then
7340        d = node.new(14, 0)   -- (penalty, userpenalty)
7341        d.attr = item_base.attr
7342        d.penalty = tovalue(crep.penalty)
7343        head, new = node.insert_before(head, item, d)
7344
7345      elseif crep and crep.space then
7346        -- 655360 = 10 pt = 10 * 65536 sp
7347        d = node.new(12, 13)      -- (glue, spaceskip)
7348        local quad = font.getfont(item_base.font).size or 655360
7349        node.setglue(d, tovalue(crep.space[1]) * quad,
7350                        tovalue(crep.space[2]) * quad,
7351                        tovalue(crep.space[3]) * quad)
7352        if mode == 0 then
7353          placeholder = ' '
7354        end
7355        head, new = node.insert_before(head, item, d)
7356
7357      elseif crep and crep.norule then
7358        -- 655360 = 10 pt = 10 * 65536 sp
7359        d = node.new(2, 3)        -- (rule, empty) = \no*rule
7360        local quad = font.getfont(item_base.font).size or 655360
7361        d.width   = tovalue(crep.norule[1]) * quad
7362        d.height  = tovalue(crep.norule[2]) * quad
7363        d.depth   = tovalue(crep.norule[3]) * quad
7364        head, new = node.insert_before(head, item, d)
7365
7366      elseif crep and crep.spacefactor then
7367        d = node.new(12, 13)       -- (glue, spaceskip)
7368        local base_font = font.getfont(item_base.font)
7369        node.setglue(d,
```

```
7370              tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7371              tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7372              tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7373            if mode == 0 then
7374              placeholder = ' '
7375            end
7376            head, new = node.insert_before(head, item, d)
7377
7378          elseif mode == 0 and crep and crep.space then
7379            -- ERROR
7380
7381          elseif crep and crep.kern then
7382            d = node.new(13, 1)      -- (kern, user)
7383            local quad = font.getfont(item_base.font).size or 655360
7384            d.attr = item_base.attr
7385            d.kern = tovalue(crep.kern) * quad
7386            head, new = node.insert_before(head, item, d)
7387
7388          elseif crep and crep.node then
7389            d = node.new(crep.node[1], crep.node[2])
7390            d.attr = item_base.attr
7391            head, new = node.insert_before(head, item, d)
7392
7393          end  -- ie replacement cases
7394
7395          -- Shared by disc, space(factor), kern, node and penalty.
7396          if sc == 1 then
7397            word_head = head
7398          end
7399          if crep.insert then
7400            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7401            table.insert(w_nodes, sc, new)
7402            last = last + 1
7403          else
7404            w_nodes[sc] = d
7405            node.remove(head, item)
7406            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7407          end
7408
7409          last_match = utf8.offset(w, sc+1+step)
7410
7411          ::next::
7412
7413        end  -- for each replacement
7414
7415        if Babel.debug then
7416            print('.....', '/')
7417            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7418        end
7419
7420      if dummy_node then
7421        node.remove(head, dummy_node)
7422        dummy_node = nil
7423      end
7424
7425      end  -- for match
7426
7427    end  -- for patterns
7428
7429    ::next::
7430    word_head = nw
7431  end  -- for substring
7432  return head
```

```
7433 end
7434
7435 -- This table stores capture maps, numbered consecutively
7436 Babel.capture_maps = {}
7437
7438 -- The following functions belong to the next macro
7439 function Babel.capture_func(key, cap)
7440   local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
7441   local cnt
7442   local u = unicode.utf8
7443   ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
7444   if cnt == 0 then
7445     ret = u.gsub(ret, '{(%x%x%x%x+)}',
7446           function (n)
7447             return u.char(tonumber(n, 16))
7448           end)
7449   end
7450   ret = ret:gsub("%[%[%]%]%.%.", '')
7451   ret = ret:gsub("%.%.%[%[%]%]", '')
7452   return key .. [[=function(m) return ]] .. ret .. [[ end]]
7453 end
7454
7455 function Babel.capt_map(from, mapno)
7456   return Babel.capture_maps[mapno][from] or from
7457 end
7458
7459 -- Handle the {n|abc|ABC} syntax in captures
7460 function Babel.capture_func_map(capno, from, to)
7461   local u = unicode.utf8
7462   from = u.gsub(from, '{(%x%x%x%x+)}',
7463         function (n)
7464           return u.char(tonumber(n, 16))
7465         end)
7466   to = u.gsub(to, '{(%x%x%x%x+)}',
7467         function (n)
7468           return u.char(tonumber(n, 16))
7469         end)
7470   local froms = {}
7471   for s in string.utfcharacters(from) do
7472     table.insert(froms, s)
7473   end
7474   local cnt = 1
7475   table.insert(Babel.capture_maps, {})
7476   local mlen = table.getn(Babel.capture_maps)
7477   for s in string.utfcharacters(to) do
7478     Babel.capture_maps[mlen][froms[cnt]] = s
7479     cnt = cnt + 1
7480   end
7481   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
7482         (mlen) .. ")..".. "[["
7483 end
7484
7485 -- Create/Extend reversed sorted list of kashida weights:
7486 function Babel.capture_kashida(key, wt)
7487   wt = tonumber(wt)
7488   if Babel.kashida_wts then
7489     for p, q in ipairs(Babel.kashida_wts) do
7490       if wt  == q then
7491         break
7492       elseif wt > q then
7493         table.insert(Babel.kashida_wts, p, wt)
7494         break
7495       elseif table.getn(Babel.kashida_wts) == p then
```

```
7496        table.insert(Babel.kashida_wts, wt)
7497      end
7498    end
7499  else
7500    Babel.kashida_wts = { wt }
7501  end
7502  return 'kashida = ' .. wt
7503 end
7504
7505 function Babel.capture_node(id, subtype)
7506  local sbt = 0
7507  for k, v in pairs(node.subtypes(id)) do
7508    if v == subtype then sbt = k end
7509  end
7510  return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7511 end
7512
7513 -- Experimental: applies prehyphenation transforms to a string (letters
7514 -- and spaces).
7515 function Babel.string_prehyphenation(str, locale)
7516  local n, head, last, res
7517  head = node.new(8, 0) -- dummy (hack just to start)
7518  last = head
7519  for s in string.utfvalues(str) do
7520    if s == 20 then
7521      n = node.new(12, 0)
7522    else
7523      n = node.new(29, 0)
7524      n.char = s
7525    end
7526    node.set_attribute(n, Babel.attr_locale, locale)
7527    last.next = n
7528    last = n
7529  end
7530  head = Babel.hyphenate_replace(head, 0)
7531  res = ''
7532  for n in node.traverse(head) do
7533    if n.id == 12 then
7534      res = res .. ' '
7535    elseif n.id == 29 then
7536      res = res .. unicode.utf8.char(n.char)
7537    end
7538  end
7539  tex.print(res)
7540 end
7541 ⟨/transforms⟩
```

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

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

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

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

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

> 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.

```
7542 ⟨∗basic-r⟩
7543 Babel.bidi_enabled = true
7544
7545 require('babel-data-bidi.lua')
7546
7547 local characters = Babel.characters
7548 local ranges = Babel.ranges
7549
7550 local DIR = node.id("dir")
7551
7552 local function dir_mark(head, from, to, outer)
7553   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7554   local d = node.new(DIR)
7555   d.dir = '+' .. dir
7556   node.insert_before(head, from, d)
7557   d = node.new(DIR)
7558   d.dir = '-' .. dir
7559   node.insert_after(head, to, d)
7560 end
7561
7562 function Babel.bidi(head, ispar)
7563   local first_n, last_n          -- first and last char with nums
7564   local last_es                  -- an auxiliary 'last' used with nums
7565   local first_d, last_d          -- first and last char in L/R block
7566   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):

```
7567   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7568   local strong_lr = (strong == 'l') and 'l' or 'r'
7569   local outer = strong
7570
7571   local new_dir = false
7572   local first_dir = false
7573   local inmath = false
7574
7575   local last_lr
7576
7577   local type_n = ''
7578
```

```
7579  for item in node.traverse(head) do
7580
7581    -- three cases: glyph, dir, otherwise
7582    if item.id == node.id'glyph'
7583      or (item.id == 7 and item.subtype == 2) then
7584
7585      local itemchar
7586      if item.id == 7 and item.subtype == 2 then
7587        itemchar = item.replace.char
7588      else
7589        itemchar = item.char
7590      end
7591      local chardata = characters[itemchar]
7592      dir = chardata and chardata.d or nil
7593      if not dir then
7594        for nn, et in ipairs(ranges) do
7595          if itemchar < et[1] then
7596            break
7597          elseif itemchar <= et[2] then
7598            dir = et[3]
7599            break
7600          end
7601        end
7602      end
7603      dir = dir or 'l'
7604      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).

```
7605      if new_dir then
7606        attr_dir = 0
7607        for at in node.traverse(item.attr) do
7608          if at.number == Babel.attr_dir then
7609            attr_dir = at.value & 0x3
7610          end
7611        end
7612        if attr_dir == 1 then
7613          strong = 'r'
7614        elseif attr_dir == 2 then
7615          strong = 'al'
7616        else
7617          strong = 'l'
7618        end
7619        strong_lr = (strong == 'l') and 'l' or 'r'
7620        outer = strong_lr
7621        new_dir = false
7622      end
7623
7624      if dir == 'nsm' then dir = strong end                 -- W1
```

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

```
7625      dir_real = dir             -- We need dir_real to set strong below
7626      if dir == 'al' then dir = 'r' end -- W3
```

By W2, there are no <en> <et> <es> if strong $== \langle al \rangle$, only <an>. Therefore, there are not <et en> nor <en et>, W5 can be ignored, and W6 applied:

```
7627      if strong == 'al' then
7628        if dir == 'en' then dir = 'an' end                 -- W2
7629        if dir == 'et' or dir == 'es' then dir = 'on' end  -- W6
7630        strong_lr = 'r'                                    -- W3
7631      end
```

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

```
7632    elseif item.id == node.id'dir' and not inmath then
7633      new_dir = true
7634      dir = nil
7635    elseif item.id == node.id'math' then
7636      inmath = (item.subtype == 0)
7637    else
7638      dir = nil              -- Not a char
7639    end
```

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

```
7640    if dir == 'en' or dir == 'an' or dir == 'et' then
7641      if dir ~= 'et' then
7642        type_n = dir
7643      end
7644      first_n = first_n or item
7645      last_n = last_es or item
7646      last_es = nil
7647    elseif dir == 'es' and last_n then -- W3+W6
7648      last_es = item
7649    elseif dir == 'cs' then            -- it's right - do nothing
7650    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7651      if strong_lr == 'r' and type_n ~= '' then
7652        dir_mark(head, first_n, last_n, 'r')
7653      elseif strong_lr == 'l' and first_d and type_n == 'an' then
7654        dir_mark(head, first_n, last_n, 'r')
7655        dir_mark(head, first_d, last_d, outer)
7656        first_d, last_d = nil, nil
7657      elseif strong_lr == 'l' and type_n ~= '' then
7658        last_d = last_n
7659      end
7660      type_n = ''
7661      first_n, last_n = nil, nil
7662    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:

```
7663    if dir == 'l' or dir == 'r' then
7664      if dir ~= outer then
7665        first_d = first_d or item
7666        last_d = item
7667      elseif first_d and dir ~= strong_lr then
7668        dir_mark(head, first_d, last_d, outer)
7669        first_d, last_d = nil, nil
7670      end
7671    end
```

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

```
7672    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7673      item.char = characters[item.char] and
7674                  characters[item.char].m or item.char
7675    elseif (dir or new_dir) and last_lr ~= item then
7676      local mir = outer .. strong_lr .. (dir or outer)
```

```
7677        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7678          for ch in node.traverse(node.next(last_lr)) do
7679            if ch == item then break end
7680            if ch.id == node.id'glyph' and characters[ch.char] then
7681              ch.char = characters[ch.char].m or ch.char
7682            end
7683          end
7684        end
7685      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`).

```
7686      if dir == 'l' or dir == 'r' then
7687        last_lr = item
7688        strong = dir_real             -- Don't search back - best save now
7689        strong_lr = (strong == 'l') and 'l' or 'r'
7690      elseif new_dir then
7691        last_lr = nil
7692      end
7693    end
```

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

```
7694    if last_lr and outer == 'r' then
7695      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7696        if characters[ch.char] then
7697          ch.char = characters[ch.char].m or ch.char
7698        end
7699      end
7700    end
7701    if first_n then
7702      dir_mark(head, first_n, last_n, outer)
7703    end
7704    if first_d then
7705      dir_mark(head, first_d, last_d, outer)
7706    end
```

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

```
7707    return node.prev(head) or head
7708 end
```
7709 ⟨/basic-r⟩

And here the Lua code for `bidi=basic`:

7710 ⟨∗basic⟩
```
7711 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7712
7713 Babel.fontmap = Babel.fontmap or {}
7714 Babel.fontmap[0] = {}        -- l
7715 Babel.fontmap[1] = {}        -- r
7716 Babel.fontmap[2] = {}        -- al/an
7717
7718 -- To cancel mirroring. Also OML, OMS, U?
7719 Babel.symbol_fonts = Babel.symbol_fonts or {}
7720 Babel.symbol_fonts[font.id('tenln')] = true
7721 Babel.symbol_fonts[font.id('tenlnw')] = true
7722 Babel.symbol_fonts[font.id('tencirc')] = true
7723 Babel.symbol_fonts[font.id('tencircw')] = true
7724
7725 Babel.bidi_enabled = true
7726 Babel.mirroring_enabled = true
7727
7728 require('babel-data-bidi.lua')
7729
7730 local characters = Babel.characters
```

155

```lua
local ranges = Babel.ranges

local DIR = node.id('dir')
local GLYPH = node.id('glyph')

local function insert_implicit(head, state, outer)
  local new_state = state
  if state.sim and state.eim and state.sim ~= state.eim then
    dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
    local d = node.new(DIR)
    d.dir = '+' .. dir
    node.insert_before(head, state.sim, d)
    local d = node.new(DIR)
    d.dir = '-' .. dir
    node.insert_after(head, state.eim, d)
  end
  new_state.sim, new_state.eim = nil, nil
  return head, new_state
end

local function insert_numeric(head, state)
  local new
  local new_state = state
  if state.san and state.ean and state.san ~= state.ean then
    local d = node.new(DIR)
    d.dir = '+TLT'
    _, new = node.insert_before(head, state.san, d)
    if state.san == state.sim then state.sim = new end
    local d = node.new(DIR)
    d.dir = '-TLT'
    _, new = node.insert_after(head, state.ean, d)
    if state.ean == state.eim then state.eim = new end
  end
  new_state.san, new_state.ean = nil, nil
  return head, new_state
end

local function glyph_not_symbol_font(node)
  if node.id == GLYPH then
    return not Babel.symbol_fonts[node.font]
  else
    return false
  end
end

-- TODO - \hbox with an explicit dir can lead to wrong results
-- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
-- was made to improve the situation, but the problem is the 3-dir
-- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
-- well.

function Babel.bidi(head, ispar, hdir)
  local d    -- d is used mainly for computations in a loop
  local prev_d = ''
  local new_d = false

  local nodes = {}
  local outer_first = nil
  local inmath = false

  local glue_d = nil
  local glue_i = nil

```

```
7794    local has_en = false
7795    local first_et = nil
7796
7797    local has_hyperlink = false
7798
7799    local ATDIR = Babel.attr_dir
7800    local attr_d
7801
7802    local save_outer
7803    local temp = node.get_attribute(head, ATDIR)
7804    if temp then
7805      temp = temp & 0x3
7806      save_outer = (temp == 0 and 'l') or
7807                   (temp == 1 and 'r') or
7808                   (temp == 2 and 'al')
7809    elseif ispar then              -- Or error? Shouldn't happen
7810      save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7811    else                          -- Or error? Shouldn't happen
7812      save_outer = ('TRT' == hdir) and 'r' or 'l'
7813    end
7814      -- when the callback is called, we are just _after_ the box,
7815      -- and the textdir is that of the surrounding text
7816    -- if not ispar and hdir ~= tex.textdir then
7817    --    save_outer = ('TRT' == hdir) and 'r' or 'l'
7818    -- end
7819    local outer = save_outer
7820    local last = outer
7821    -- 'al' is only taken into account in the first, current loop
7822    if save_outer == 'al' then save_outer = 'r' end
7823
7824    local fontmap = Babel.fontmap
7825
7826    for item in node.traverse(head) do
7827
7828      -- In what follows, #node is the last (previous) node, because the
7829      -- current one is not added until we start processing the neutrals.
7830
7831      -- three cases: glyph, dir, otherwise
7832      if glyph_not_symbol_font(item)
7833         or (item.id == 7 and item.subtype == 2) then
7834
7835        if node.get_attribute(item, ATDIR) == 128 then goto nextnode end
7836
7837        local d_font = nil
7838        local item_r
7839        if item.id == 7 and item.subtype == 2 then
7840          item_r = item.replace   -- automatic discs have just 1 glyph
7841        else
7842          item_r = item
7843        end
7844
7845        local chardata = characters[item_r.char]
7846        d = chardata and chardata.d or nil
7847        if not d or d == 'nsm' then
7848          for nn, et in ipairs(ranges) do
7849            if item_r.char < et[1] then
7850              break
7851            elseif item_r.char <= et[2] then
7852              if not d then d = et[3]
7853              elseif d == 'nsm' then d_font = et[3]
7854              end
7855              break
7856            end
```

```
7857            end
7858          end
7859          d = d or 'l'
7860
7861          -- A short 'pause' in bidi for mapfont
7862          d_font = d_font or d
7863          d_font = (d_font == 'l' and 0) or
7864                   (d_font == 'nsm' and 0) or
7865                   (d_font == 'r' and 1) or
7866                   (d_font == 'al' and 2) or
7867                   (d_font == 'an' and 2) or nil
7868          if d_font and fontmap and fontmap[d_font][item_r.font] then
7869            item_r.font = fontmap[d_font][item_r.font]
7870          end
7871
7872          if new_d then
7873            table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7874            if inmath then
7875              attr_d = 0
7876            else
7877              attr_d = node.get_attribute(item, ATDIR)
7878              attr_d = attr_d & 0x3
7879            end
7880            if attr_d == 1 then
7881              outer_first = 'r'
7882              last = 'r'
7883            elseif attr_d == 2 then
7884              outer_first = 'r'
7885              last = 'al'
7886            else
7887              outer_first = 'l'
7888              last = 'l'
7889            end
7890            outer = last
7891            has_en = false
7892            first_et = nil
7893            new_d = false
7894          end
7895
7896          if glue_d then
7897            if (d == 'l' and 'l' or 'r') ~= glue_d then
7898              table.insert(nodes, {glue_i, 'on', nil})
7899            end
7900            glue_d = nil
7901            glue_i = nil
7902          end
7903
7904      elseif item.id == DIR then
7905        d = nil
7906
7907        if head ~= item then new_d = true end
7908
7909      elseif item.id == node.id'glue' and item.subtype == 13 then
7910        glue_d = d
7911        glue_i = item
7912        d = nil
7913
7914      elseif item.id == node.id'math' then
7915        inmath = (item.subtype == 0)
7916
7917      elseif item.id == 8 and item.subtype == 19 then
7918        has_hyperlink = true
7919
```

```
7920    else
7921      d = nil
7922    end
7923
7924    -- AL <= EN/ET/ES     -- W2 + W3 + W6
7925    if last == 'al' and d == 'en' then
7926      d = 'an'          -- W3
7927    elseif last == 'al' and (d == 'et' or d == 'es') then
7928      d = 'on'          -- W6
7929    end
7930
7931    -- EN + CS/ES + EN    -- W4
7932    if d == 'en' and #nodes >= 2 then
7933      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7934          and nodes[#nodes-1][2] == 'en' then
7935        nodes[#nodes][2] = 'en'
7936      end
7937    end
7938
7939    -- AN + CS + AN       -- W4 too, because uax9 mixes both cases
7940    if d == 'an' and #nodes >= 2 then
7941      if (nodes[#nodes][2] == 'cs')
7942          and nodes[#nodes-1][2] == 'an' then
7943        nodes[#nodes][2] = 'an'
7944      end
7945    end
7946
7947    -- ET/EN              -- W5 + W7->l / W6->on
7948    if d == 'et' then
7949      first_et = first_et or (#nodes + 1)
7950    elseif d == 'en' then
7951      has_en = true
7952      first_et = first_et or (#nodes + 1)
7953    elseif first_et then      -- d may be nil here !
7954      if has_en then
7955        if last == 'l' then
7956          temp = 'l'     -- W7
7957        else
7958          temp = 'en'    -- W5
7959        end
7960      else
7961        temp = 'on'      -- W6
7962      end
7963      for e = first_et, #nodes do
7964        if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7965      end
7966      first_et = nil
7967      has_en = false
7968    end
7969
7970    -- Force mathdir in math if ON (currently works as expected only
7971    -- with 'l')
7972
7973    if inmath and d == 'on' then
7974      d = ('TRT' == tex.mathdir) and 'r' or 'l'
7975    end
7976
7977    if d then
7978      if d == 'al' then
7979        d = 'r'
7980        last = 'al'
7981      elseif d == 'l' or d == 'r' then
7982        last = d
```

```
7983        end
7984        prev_d = d
7985        table.insert(nodes, {item, d, outer_first})
7986      end
7987
7988      node.set_attribute(item, ATDIR, 128)
7989      outer_first = nil
7990
7991      ::nextnode::
7992
7993  end -- for each node
7994
7995  -- TODO -- repeated here in case EN/ET is the last node. Find a
7996  -- better way of doing things:
7997  if first_et then        -- dir may be nil here !
7998    if has_en then
7999      if last == 'l' then
8000        temp = 'l'     -- W7
8001      else
8002        temp = 'en'    -- W5
8003      end
8004    else
8005      temp = 'on'      -- W6
8006    end
8007    for e = first_et, #nodes do
8008      if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8009    end
8010  end
8011
8012  -- dummy node, to close things
8013  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8014
8015  --------------  NEUTRAL -----------------
8016
8017  outer = save_outer
8018  last = outer
8019
8020  local first_on = nil
8021
8022  for q = 1, #nodes do
8023    local item
8024
8025    local outer_first = nodes[q][3]
8026    outer = outer_first or outer
8027    last = outer_first or last
8028
8029    local d = nodes[q][2]
8030    if d == 'an' or d == 'en' then d = 'r' end
8031    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8032
8033    if d == 'on' then
8034      first_on = first_on or q
8035    elseif first_on then
8036      if last == d then
8037        temp = d
8038      else
8039        temp = outer
8040      end
8041      for r = first_on, q - 1 do
8042        nodes[r][2] = temp
8043        item = nodes[r][1]    -- MIRRORING
8044        if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8045             and temp == 'r' and characters[item.char] then
```

```
8046          local font_mode = ''
8047          if item.font > 0 and font.fonts[item.font].properties then
8048            font_mode = font.fonts[item.font].properties.mode
8049          end
8050          if font_mode ~= 'harf' and font_mode ~= 'plug' then
8051            item.char = characters[item.char].m or item.char
8052          end
8053        end
8054      end
8055      first_on = nil
8056    end
8057
8058    if d == 'r' or d == 'l' then last = d end
8059 end
8060
8061 --------------  IMPLICIT, REORDER ----------------
8062
8063 outer = save_outer
8064 last = outer
8065
8066 local state = {}
8067 state.has_r = false
8068
8069 for q = 1, #nodes do
8070
8071    local item = nodes[q][1]
8072
8073    outer = nodes[q][3] or outer
8074
8075    local d = nodes[q][2]
8076
8077    if d == 'nsm' then d = last end              -- W1
8078    if d == 'en' then d = 'an' end
8079    local isdir = (d == 'r' or d == 'l')
8080
8081    if outer == 'l' and d == 'an' then
8082      state.san = state.san or item
8083      state.ean = item
8084    elseif state.san then
8085      head, state = insert_numeric(head, state)
8086    end
8087
8088    if outer == 'l' then
8089      if d == 'an' or d == 'r' then     -- im -> implicit
8090        if d == 'r' then state.has_r = true end
8091        state.sim = state.sim or item
8092        state.eim = item
8093      elseif d == 'l' and state.sim and state.has_r then
8094        head, state = insert_implicit(head, state, outer)
8095      elseif d == 'l' then
8096        state.sim, state.eim, state.has_r = nil, nil, false
8097      end
8098    else
8099      if d == 'an' or d == 'l' then
8100        if nodes[q][3] then -- nil except after an explicit dir
8101          state.sim = item  -- so we move sim 'inside' the group
8102        else
8103          state.sim = state.sim or item
8104        end
8105        state.eim = item
8106      elseif d == 'r' and state.sim then
8107        head, state = insert_implicit(head, state, outer)
8108      elseif d == 'r' then
```

161

```
8109          state.sim, state.eim = nil, nil
8110       end
8111    end
8112
8113    if isdir then
8114      last = d             -- Don't search back - best save now
8115    elseif d == 'on' and state.san  then
8116      state.san = state.san or item
8117      state.ean = item
8118    end
8119
8120  end
8121
8122  head = node.prev(head) or head
8123
8124  -------------- FIX HYPERLINKS ----------------
8125
8126  if has_hyperlink then
8127    local flag, linking = 0, 0
8128    for item in node.traverse(head) do
8129      if item.id == DIR then
8130        if item.dir == '+TRT' or item.dir == '+TLT' then
8131          flag = flag + 1
8132        elseif item.dir == '-TRT' or item.dir == '-TLT' then
8133          flag = flag - 1
8134        end
8135      elseif item.id == 8 and item.subtype == 19 then
8136        linking = flag
8137      elseif item.id == 8 and item.subtype == 20 then
8138        if linking > 0 then
8139          if item.prev.id == DIR and
8140              (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8141            d = node.new(DIR)
8142            d.dir = item.prev.dir
8143            node.remove(head, item.prev)
8144            node.insert_after(head, item, d)
8145          end
8146        end
8147        linking = 0
8148      end
8149    end
8150  end
8151
8152  return head
8153 end
8154 -- Make sure anything is marked as 'bidi done' (including nodes inserted
8155 -- after the babel algorithm).
8156 function Babel.unset_atdir(head)
8157   local ATDIR = Babel.attr_dir
8158   for item in node.traverse(head) do
8159     node.set_attribute(item, ATDIR, 128)
8160   end
8161   return head
8162 end
8163 ⟨/basic⟩
```

11. Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
% [0x0021]={c='ex'},
% [0x0024]={c='pr'},
```

```
%   [0x0025]={c='po'},
%   [0x0028]={c='op'},
%   [0x0029]={c='cp'},
%   [0x002B]={c='pr'},
%
```

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

12. The 'nil' language

This 'language' does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the @ sign, etc.

8164 ⟨*nil⟩
8165 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8166 \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.

8167 \ifx\l@nil\@undefined
8168   \newlanguage\l@nil
8169   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
8170   \let\bbl@elt\relax
8171   \edef\bbl@languages{% Add it to the list of languages
8172   \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
8173 \fi

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

8174 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

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

\captionnil
\datenil

8175 \let\captionsnil\@empty
8176 \let\datenil\@empty

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

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

```
8199 \@namedef{bbl@elname@nil}{nil}
8200 \@namedef{bbl@lname@nil}{nil}
8201 \@namedef{bbl@esname@nil}{Latin}
8202 \@namedef{bbl@sname@nil}{Latin}
8203 \@namedef{bbl@sbcp@nil}{Latn}
8204 \@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.

```
8205 \ldf@finish{nil}
8206 ⟨/nil⟩
```

13. Calendars

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

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

```
8207 ⟨⟨*Compute Julian day⟩⟩ ≡
8208 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
8209 \def\bbl@cs@gregleap#1{%
8210   (\bbl@fpmod{#1}{4} == 0) &&
8211   (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
8212 \def\bbl@cs@jd#1#2#3{% year, month, day
8213   \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
8214   floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
8215   floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
8216   ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
8217 ⟨⟨/Compute Julian day⟩⟩
```

13.1. Islamic

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

```
8218 ⟨*ca-islamic⟩
8219 \ExplSyntaxOn
8220 <@Compute Julian day@>
8221 % == islamic (default)
8222 % Not yet implemented
8223 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
8224 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8225   ((#3 + ceil(29.5 * (#2 - 1)) +
8226   (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8227   1948439.5) - 1) }
8228 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8229 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8230 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8231 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8232 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8233 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
8234   \edef\bbl@tempa{%
8235     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8236   \edef#5{%
8237     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8238   \edef#6{\fp_eval:n{
8239     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8240   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}
```

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

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

```
8241 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8242   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8243   57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8244   57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8245   57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8246   58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8247   58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8248   58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8249   58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8250   59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8251   59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8252   59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8253   60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8254   60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8255   60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8256   60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8257   61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8258   61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8259   61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8260   62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8261   62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8262   62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8263   63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8264   63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8265   63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8266   63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8267   64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8268   64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8269   64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8270   65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8271   65401,65431,65460,65490,65520}
8272 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8273 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8274 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8275 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
8276   \ifnum#2>2014 \ifnum#2<2038
8277     \bbl@afterfi\expandafter\@gobble
8278   \fi\fi
8279     {\bbl@error{year-out-range}{2014-2038}{}{}}%
8280   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8281     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8282   \count@\@ne
8283   \bbl@foreach\bbl@cs@umalqura@data{%
8284     \advance\count@\@ne
8285     \ifnum##1>\bbl@tempd\else
8286       \edef\bbl@tempe{\the\count@}%
8287       \edef\bbl@tempb{##1}%
8288     \fi}%
8289   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8290   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8291   \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
8292   \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8293   \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
8294 \ExplSyntaxOff
8295 \bbl@add\bbl@precalendar{%
8296   \bbl@replace\bbl@ld@calendar{-civil}{}%
8297   \bbl@replace\bbl@ld@calendar{-umalqura}{}%
8298   \bbl@replace\bbl@ld@calendar{+}{}%
8299   \bbl@replace\bbl@ld@calendar{-}{}}
8300 ⟨/ca-islamic⟩
```

13.2. Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptions 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`

```
8301 ⟨∗ca-hebrew⟩
8302 \newcount\bbl@cntcommon
8303 \def\bbl@remainder#1#2#3{%
8304   #3=#1\relax
8305   \divide #3 by #2\relax
8306   \multiply #3 by -#2\relax
8307   \advance #3 by #1\relax}%
8308 \newif\ifbbl@divisible
8309 \def\bbl@checkifdivisible#1#2{%
8310   {\countdef\tmp=0
8311    \bbl@remainder{#1}{#2}{\tmp}%
8312    \ifnum \tmp=0
8313        \global\bbl@divisibletrue
8314    \else
8315        \global\bbl@divisiblefalse
8316    \fi}}
8317 \newif\ifbbl@gregleap
8318 \def\bbl@ifgregleap#1{%
8319   \bbl@checkifdivisible{#1}{4}%
8320   \ifbbl@divisible
8321       \bbl@checkifdivisible{#1}{100}%
8322       \ifbbl@divisible
8323           \bbl@checkifdivisible{#1}{400}%
8324           \ifbbl@divisible
8325               \bbl@gregleaptrue
8326           \else
8327               \bbl@gregleapfalse
8328           \fi
8329       \else
8330           \bbl@gregleaptrue
8331       \fi
8332   \else
8333       \bbl@gregleapfalse
8334   \fi
8335   \ifbbl@gregleap}
8336 \def\bbl@gregdayspriormonths#1#2#3{%
8337     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8338         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8339     \bbl@ifgregleap{#2}%
8340         \ifnum #1 > 2
8341             \advance #3 by 1
8342         \fi
8343     \fi
8344     \global\bbl@cntcommon=#3}%
8345     #3=\bbl@cntcommon}
8346 \def\bbl@gregdaysprioryears#1#2{%
8347   {\countdef\tmpc=4
8348    \countdef\tmpb=2
8349    \tmpb=#1\relax
8350    \advance \tmpb by -1
8351    \tmpc=\tmpb
8352    \multiply \tmpc by 365
8353    #2=\tmpc
8354    \tmpc=\tmpb
8355    \divide \tmpc by 4
8356    \advance #2 by \tmpc
8357    \tmpc=\tmpb
8358    \divide \tmpc by 100
```

```
8359    \advance #2 by -\tmpc
8360    \tmpc=\tmpb
8361    \divide \tmpc by 400
8362    \advance #2 by \tmpc
8363    \global\bbl@cntcommon=#2\relax}%
8364    #2=\bbl@cntcommon}
8365 \def\bbl@absfromgreg#1#2#3#4{%
8366    {\countdef\tmpd=0
8367    #4=#1\relax
8368    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8369    \advance #4 by \tmpd
8370    \bbl@gregdaysprioryears{#3}{\tmpd}%
8371    \advance #4 by \tmpd
8372    \global\bbl@cntcommon=#4\relax}%
8373    #4=\bbl@cntcommon}
8374 \newif\ifbbl@hebrleap
8375 \def\bbl@checkleaphebryear#1{%
8376    {\countdef\tmpa=0
8377    \countdef\tmpb=1
8378    \tmpa=#1\relax
8379    \multiply \tmpa by 7
8380    \advance \tmpa by 1
8381    \bbl@remainder{\tmpa}{19}{\tmpb}%
8382    \ifnum \tmpb < 7
8383        \global\bbl@hebrleaptrue
8384    \else
8385        \global\bbl@hebrleapfalse
8386    \fi}}
8387 \def\bbl@hebrelapsedmonths#1#2{%
8388    {\countdef\tmpa=0
8389    \countdef\tmpb=1
8390    \countdef\tmpc=2
8391    \tmpa=#1\relax
8392    \advance \tmpa by -1
8393    #2=\tmpa
8394    \divide #2 by 19
8395    \multiply #2 by 235
8396    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8397    \tmpc=\tmpb
8398    \multiply \tmpb by 12
8399    \advance #2 by \tmpb
8400    \multiply \tmpc by 7
8401    \advance \tmpc by 1
8402    \divide \tmpc by 19
8403    \advance #2 by \tmpc
8404    \global\bbl@cntcommon=#2}%
8405    #2=\bbl@cntcommon}
8406 \def\bbl@hebrelapseddays#1#2{%
8407    {\countdef\tmpa=0
8408    \countdef\tmpb=1
8409    \countdef\tmpc=2
8410    \bbl@hebrelapsedmonths{#1}{#2}%
8411    \tmpa=#2\relax
8412    \multiply \tmpa by 13753
8413    \advance \tmpa by 5604
8414    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8415    \divide \tmpa by 25920
8416    \multiply #2 by 29
8417    \advance #2 by 1
8418    \advance #2 by \tmpa
8419    \bbl@remainder{#2}{7}{\tmpa}%
8420    \ifnum \tmpc < 19440
8421        \ifnum \tmpc < 9924
```

```
8422        \else
8423            \ifnum \tmpa=2
8424                \bbl@checkleaphebryear{#1}% of a common year
8425                \ifbbl@hebrleap
8426                \else
8427                    \advance #2 by 1
8428                \fi
8429            \fi
8430        \fi
8431        \ifnum \tmpc < 16789
8432        \else
8433            \ifnum \tmpa=1
8434                \advance #1 by -1
8435                \bbl@checkleaphebryear{#1}% at the end of leap year
8436                \ifbbl@hebrleap
8437                    \advance #2 by 1
8438                \fi
8439            \fi
8440        \fi
8441    \else
8442        \advance #2 by 1
8443    \fi
8444    \bbl@remainder{#2}{7}{\tmpa}%
8445    \ifnum \tmpa=0
8446        \advance #2 by 1
8447    \else
8448        \ifnum \tmpa=3
8449            \advance #2 by 1
8450        \else
8451            \ifnum \tmpa=5
8452                \advance #2 by 1
8453            \fi
8454        \fi
8455    \fi
8456    \global\bbl@cntcommon=#2\relax}%
8457  #2=\bbl@cntcommon}
8458 \def\bbl@daysinhebryear#1#2{%
8459  {\countdef\tmpe=12
8460    \bbl@hebrelapseddays{#1}{\tmpe}%
8461    \advance #1 by 1
8462    \bbl@hebrelapseddays{#1}{#2}%
8463    \advance #2 by -\tmpe
8464    \global\bbl@cntcommon=#2}%
8465  #2=\bbl@cntcommon}
8466 \def\bbl@hebrdayspriormonths#1#2#3{%
8467  {\countdef\tmpf= 14
8468    #3=\ifcase #1
8469          0 \or
8470          0 \or
8471         30 \or
8472         59 \or
8473         89 \or
8474        118 \or
8475        148 \or
8476        148 \or
8477        177 \or
8478        207 \or
8479        236 \or
8480        266 \or
8481        295 \or
8482        325 \or
8483        400
8484    \fi
```

```
8485    \bbl@checkleaphebryear{#2}%
8486    \ifbbl@hebrleap
8487        \ifnum #1 > 6
8488            \advance #3 by 30
8489        \fi
8490    \fi
8491    \bbl@daysinhebryear{#2}{\tmpf}%
8492    \ifnum #1 > 3
8493        \ifnum \tmpf=353
8494            \advance #3 by -1
8495        \fi
8496        \ifnum \tmpf=383
8497            \advance #3 by -1
8498        \fi
8499    \fi
8500    \ifnum #1 > 2
8501        \ifnum \tmpf=355
8502            \advance #3 by 1
8503        \fi
8504        \ifnum \tmpf=385
8505            \advance #3 by 1
8506        \fi
8507    \fi
8508    \global\bbl@cntcommon=#3\relax}%
8509    #3=\bbl@cntcommon}
8510 \def\bbl@absfromhebr#1#2#3#4{%
8511    {#4=#1\relax
8512    \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8513    \advance #4 by #1\relax
8514    \bbl@hebrelapseddays{#3}{#1}%
8515    \advance #4 by #1\relax
8516    \advance #4 by -1373429
8517    \global\bbl@cntcommon=#4\relax}%
8518    #4=\bbl@cntcommon}
8519 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8520    {\countdef\tmpx= 17
8521    \countdef\tmpy= 18
8522    \countdef\tmpz= 19
8523    #6=#3\relax
8524    \global\advance #6 by 3761
8525    \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8526    \tmpz=1  \tmpy=1
8527    \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8528    \ifnum \tmpx > #4\relax
8529        \global\advance #6 by -1
8530        \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8531    \fi
8532    \advance #4 by -\tmpx
8533    \advance #4 by 1
8534    #5=#4\relax
8535    \divide #5 by 30
8536    \loop
8537        \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8538        \ifnum \tmpx < #4\relax
8539            \advance #5 by 1
8540            \tmpy=\tmpx
8541    \repeat
8542    \global\advance #5 by -1
8543    \global\advance #4 by -\tmpy}}
8544 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8545 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8546 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8547  \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
```

```
8548  \bbl@hebrfromgreg
8549    {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8550    {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8551  \edef#4{\the\bbl@hebryear}%
8552  \edef#5{\the\bbl@hebrmonth}%
8553  \edef#6{\the\bbl@hebrday}}
```
8554 ⟨/ca-hebrew⟩

13.3. Persian

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

8555 ⟨∗ca-persian⟩
```
8556 \ExplSyntaxOn
8557 <@Compute Julian day@>
8558 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8559   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8560 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8561   \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
8562   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8563     \bbl@afterfi\expandafter\@gobble
8564   \fi\fi
8565     {\bbl@error{year-out-range}{2013-2050}{}{}}%
8566   \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8567   \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8568   \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8569   \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8570   \ifnum\bbl@tempc<\bbl@tempb
8571     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8572     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8573     \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8574     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8575   \fi
8576   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8577   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8578   \edef#5{\fp_eval:n{% set Jalali month
8579     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8580   \edef#6{\fp_eval:n{% set Jalali day
8581     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}
8582 \ExplSyntaxOff
```
8583 ⟨/ca-persian⟩

13.4. Coptic and Ethiopic

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

8584 ⟨∗ca-coptic⟩
```
8585 \ExplSyntaxOn
8586 <@Compute Julian day@>
8587 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8588   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8589   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8590   \edef#4{\fp_eval:n{%
8591     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8592   \edef\bbl@tempc{\fp_eval:n{%
8593     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8594   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8595   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
8596 \ExplSyntaxOff
```

```
8597 ⟨/ca-coptic⟩
8598 ⟨∗ca-ethiopic⟩
8599 \ExplSyntaxOn
8600 <@Compute Julian day@>
8601 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8602   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8603   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8604   \edef#4{\fp_eval:n{%
8605     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8606   \edef\bbl@tempc{\fp_eval:n{%
8607     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8608   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8609   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
8610 \ExplSyntaxOff
8611 ⟨/ca-ethiopic⟩
```

13.5. Buddhist

That's very simple.

```
8612 ⟨∗ca-buddhist⟩
8613 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8614   \edef#4{\number\numexpr#1+543\relax}%
8615   \edef#5{#2}%
8616   \edef#6{#3}}
8617 ⟨/ca-buddhist⟩
8618 %
8619 % \subsection{Chinese}
8620 %
8621 % Brute force, with the Julian day of first day of each month. The
8622 % table has been computed with the help of \textsf{python-lunardate} by
8623 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8624 % is 2015-2044.
8625 %
8626 %    \begin{macrocode}
8627 ⟨∗ca-chinese⟩
8628 \ExplSyntaxOn
8629 <@Compute Julian day@>
8630 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8631   \edef\bbl@tempd{\fp_eval:n{%
8632     \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8633   \count@\z@
8634   \@tempcnta=2015
8635   \bbl@foreach\bbl@cs@chinese@data{%
8636     \ifnum##1>\bbl@tempd\else
8637       \advance\count@\@ne
8638       \ifnum\count@>12
8639         \count@\@ne
8640         \advance\@tempcnta\@ne\fi
8641       \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8642       \ifin@
8643         \advance\count@\m@ne
8644         \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8645       \else
8646         \edef\bbl@tempe{\the\count@}%
8647       \fi
8648       \edef\bbl@tempb{##1}%
8649     \fi}%
8650   \edef#4{\the\@tempcnta}%
8651   \edef#5{\bbl@tempe}%
8652   \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8653 \def\bbl@cs@chinese@leap{%
8654   885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8655 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
```

```
8656  354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8657  768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8658  1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8659  1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8660  1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8661  2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8662  2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8663  2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8664  3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8665  3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8666  3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8667  4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8668  4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8669  5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8670  5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8671  5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8672  6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8673  6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8674  6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8675  7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8676  7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8677  7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8678  8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8679  8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8680  8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8681  9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8682  9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8683  10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8684  10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8685  10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8686  10896,10926,10956,10986,11015,11045,11074,11103}
8687 \ExplSyntaxOff
8688 ⟨/ca-chinese⟩
```

14. Support for Plain TₑX (`plain.def`)

14.1. Not renaming `hyphen.tex`

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

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

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

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

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

```
8689 ⟨∗bplain | blplain⟩
8690 \catcode`\{=1 % left brace is begin-group character
8691 \catcode`\}=2 % right brace is end-group character
8692 \catcode`\#=6 % hash mark is macro parameter character
```

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

```
8693 \openin 0 hyphen.cfg
8694 \ifeof0
```

```
8695 \else
8696   \let\a\input
```

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

```
8697   \def\input #1 {%
8698     \let\input\a
8699     \a hyphen.cfg
8700     \let\a\undefined
8701   }
8702 \fi
8703 ⟨/bplain | blplain⟩
```

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

```
8704 ⟨bplain⟩\a plain.tex
8705 ⟨blplain⟩\a lplain.tex
```

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

```
8706 ⟨bplain⟩\def\fmtname{babel-plain}
8707 ⟨blplain⟩\def\fmtname{babel-lplain}
```

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

14.2. Emulating some LATEX features

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

```
8708 ⟨⟨*Emulate LaTeX⟩⟩ ≡
8709 \def\@empty{}
8710 \def\loadlocalcfg#1{%
8711   \openin0#1.cfg
8712   \ifeof0
8713     \closein0
8714   \else
8715     \closein0
8716     {\immediate\write16{***********************************}%
8717      \immediate\write16{* Local config file #1.cfg used}%
8718      \immediate\write16{*}%
8719      }
8720     \input #1.cfg\relax
8721   \fi
8722   \@endofldf}
```

14.3. General tools

A number of LATEX macro's that are needed later on.

```
8723 \long\def\@firstofone#1{#1}
8724 \long\def\@firstoftwo#1#2{#1}
8725 \long\def\@secondoftwo#1#2{#2}
8726 \def\@nnil{\@nil}
8727 \def\@gobbletwo#1#2{}
8728 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8729 \def\@star@or@long#1{%
8730   \@ifstar
8731   {\let\l@ngrel@x\relax#1}%
8732   {\let\l@ngrel@x\long#1}}
8733 \let\l@ngrel@x\relax
```

```
8734 \def\@car#1#2\@nil{#1}
8735 \def\@cdr#1#2\@nil{#2}
8736 \let\@typeset@protect\relax
8737 \let\protected@edef\edef
8738 \long\def\@gobble#1{}
8739 \edef\@backslashchar{\expandafter\@gobble\string\\}
8740 \def\strip@prefix#1>{}
8741 \def\g@addto@macro#1#2{{%
8742     \toks@\expandafter{#1#2}%
8743     \xdef#1{\the\toks@}}}
8744 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8745 \def\@nameuse#1{\csname #1\endcsname}
8746 \def\@ifundefined#1{%
8747   \expandafter\ifx\csname#1\endcsname\relax
8748     \expandafter\@firstoftwo
8749   \else
8750     \expandafter\@secondoftwo
8751   \fi}
8752 \def\@expandtwoargs#1#2#3{%
8753   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8754 \def\zap@space#1 #2{%
8755   #1%
8756   \ifx#2\@empty\else\expandafter\zap@space\fi
8757   #2}
8758 \let\bbl@trace\@gobble
8759 \def\bbl@error#1{% Implicit #2#3#4
8760   \begingroup
8761     \catcode`\\=0   \catcode`\==12 \catcode`\`=12
8762     \catcode`\^^M=5 \catcode`\%=14
8763     \input errbabel.def
8764   \endgroup
8765   \bbl@error{#1}}
8766 \def\bbl@warning#1{%
8767   \begingroup
8768     \newlinechar=`\^^J
8769     \def\\{^^J(babel) }%
8770     \message{\\#1}%
8771   \endgroup}
8772 \let\bbl@infowarn\bbl@warning
8773 \def\bbl@info#1{%
8774   \begingroup
8775     \newlinechar=`\^^J
8776     \def\\{^^J}%
8777     \wlog{#1}%
8778   \endgroup}
```

LATEX 2ε has the command \@onlypreamble which adds commands to a list of commands that are
no longer needed after \begin{document}.

```
8779 \ifx\@preamblecmds\@undefined
8780   \def\@preamblecmds{}
8781 \fi
8782 \def\@onlypreamble#1{%
8783   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8784     \@preamblecmds\do#1}}
8785 \@onlypreamble\@onlypreamble
```

Mimic LATEX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```
8786 \def\begindocument{%
8787   \@begindocumenthook
8788   \global\let\@begindocumenthook\@undefined
8789   \def\do##1{\global\let##1\@undefined}%
8790   \@preamblecmds
8791   \global\let\do\noexpand}
```

174

```
8792 \ifx\@begindocumenthook\@undefined
8793   \def\@begindocumenthook{}
8794 \fi
8795 \@onlypreamble\@begindocumenthook
8796 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
```

We also have to mimic LATEX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```
8797 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8798 \@onlypreamble\AtEndOfPackage
8799 \def\@endofldf{}
8800 \@onlypreamble\@endofldf
8801 \let\bbl@afterlang\@empty
8802 \chardef\bbl@opt@hyphenmap\z@
```

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

```
8803 \catcode`\&=\z@
8804 \ifx&if@filesw\@undefined
8805   \expandafter\let\csname if@filesw\expandafter\endcsname
8806     \csname iffalse\endcsname
8807 \fi
8808 \catcode`\&=4
```

Mimic LATEX's commands to define control sequences.

```
8809 \def\newcommand{\@star@or@long\new@command}
8810 \def\new@command#1{%
8811   \@testopt{\@newcommand#1}0}
8812 \def\@newcommand#1[#2]{%
8813   \@ifnextchar [{\@xargdef#1[#2]}%
8814                 {\@argdef#1[#2]}}
8815 \long\def\@argdef#1[#2]#3{%
8816   \@yargdef#1\@ne{#2}{#3}}
8817 \long\def\@xargdef#1[#2][#3]#4{%
8818   \expandafter\def\expandafter#1\expandafter{%
8819     \expandafter\@protected@testopt\expandafter #1%
8820     \csname\string#1\expandafter\endcsname{#3}}%
8821   \expandafter\@yargdef \csname\string#1\endcsname
8822   \tw@{#2}{#4}}
8823 \long\def\@yargdef#1#2#3{%
8824   \@tempcnta#3\relax
8825   \advance \@tempcnta \@ne
8826   \let\@hash@\relax
8827   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8828   \@tempcntb #2%
8829   \@whilenum\@tempcntb <\@tempcnta
8830   \do{%
8831     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8832     \advance\@tempcntb \@ne}%
8833   \let\@hash@##%
8834   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8835 \def\providecommand{\@star@or@long\provide@command}
8836 \def\provide@command#1{%
8837   \begingroup
8838     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
8839   \endgroup
8840   \expandafter\@ifundefined\@gtempa
8841     {\def\reserved@a{\new@command#1}}%
8842     {\let\reserved@a\relax
8843      \def\reserved@a{\new@command\reserved@a}}%
8844   \reserved@a}%

8845 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
```

```
8846 \def\declare@robustcommand#1{%
8847    \edef\reserved@a{\string#1}%
8848    \def\reserved@b{#1}%
8849    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8850    \edef#1{%
8851       \ifx\reserved@a\reserved@b
8852          \noexpand\x@protect
8853          \noexpand#1%
8854       \fi
8855       \noexpand\protect
8856       \expandafter\noexpand\csname
8857          \expandafter\@gobble\string#1 \endcsname
8858    }%
8859    \expandafter\new@command\csname
8860       \expandafter\@gobble\string#1 \endcsname
8861 }
8862 \def\x@protect#1{%
8863    \ifx\protect\@typeset@protect\else
8864       \@x@protect#1%
8865    \fi
8866 }
8867 \catcode`\&=\z@  % Trick to hide conditionals
8868    \def\@x@protect#1&fi#2#3{&fi\protect#1}
```

The following little macro \in@ is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean \in@; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \bbl@tempa.

```
8869    \def\bbl@tempa{\csname newif\endcsname&ifin@}
8870 \catcode`\&=4
8871 \ifx\in@\@undefined
8872    \def\in@#1#2{%
8873       \def\in@@##1#1##2##3\in@@{%
8874          \ifx\in@##2\in@false\else\in@true\fi}%
8875       \in@@#2#1\in@\in@@}
8876 \else
8877    \let\bbl@tempa\@empty
8878 \fi
8879 \bbl@tempa
```

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

```
8880 \def\@ifpackagewith#1#2#3#4{#3}
```

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

```
8881 \def\@ifl@aded#1#2#3#4{}
```

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

```
8882 \ifx\@tempcnta\@undefined
8883    \csname newcount\endcsname\@tempcnta\relax
8884 \fi
8885 \ifx\@tempcntb\@undefined
8886    \csname newcount\endcsname\@tempcntb\relax
8887 \fi
```

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

```
8888 \ifx\bye\@undefined
8889    \advance\count10 by -2\relax
```

176

```
8890 \fi
8891 \ifx\@ifnextchar\@undefined
8892   \def\@ifnextchar#1#2#3{%
8893     \let\reserved@d=#1%
8894     \def\reserved@a{#2}\def\reserved@b{#3}%
8895     \futurelet\@let@token\@ifnch}
8896   \def\@ifnch{%
8897     \ifx\@let@token\@sptoken
8898       \let\reserved@c\@xifnch
8899     \else
8900       \ifx\@let@token\reserved@d
8901         \let\reserved@c\reserved@a
8902       \else
8903         \let\reserved@c\reserved@b
8904       \fi
8905     \fi
8906     \reserved@c}
8907   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
8908   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
8909 \fi
8910 \def\@testopt#1#2{%
8911   \@ifnextchar[{#1}{#1[#2]}}
8912 \def\@protected@testopt#1{%
8913   \ifx\protect\@typeset@protect
8914     \expandafter\@testopt
8915   \else
8916     \@x@protect#1%
8917   \fi}
8918 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8919       #2\relax}\fi}
8920 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8921         \else\expandafter\@gobble\fi{#1}}
```

14.4. Encoding related macros

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

```
8922 \def\DeclareTextCommand{%
8923   \@dec@text@cmd\providecommand
8924 }
8925 \def\ProvideTextCommand{%
8926   \@dec@text@cmd\providecommand
8927 }
8928 \def\DeclareTextSymbol#1#2#3{%
8929   \@dec@text@cmd\chardef#1{#2}#3\relax
8930 }
8931 \def\@dec@text@cmd#1#2#3{%
8932   \expandafter\def\expandafter#2%
8933       \expandafter{%
8934         \csname#3-cmd\expandafter\endcsname
8935         \expandafter#2%
8936         \csname#3\string#2\endcsname
8937       }%
8938 %   \let\@ifdefinable\@rc@ifdefinable
8939   \expandafter#1\csname#3\string#2\endcsname
8940 }
8941 \def\@current@cmd#1{%
8942   \ifx\protect\@typeset@protect\else
8943       \noexpand#1\expandafter\@gobble
8944   \fi
8945 }
8946 \def\@changed@cmd#1#2{%
8947   \ifx\protect\@typeset@protect
8948       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
```

```
8949      \expandafter\ifx\csname ?\string#1\endcsname\relax
8950        \expandafter\def\csname ?\string#1\endcsname{%
8951          \@changed@x@err{#1}%
8952        }%
8953      \fi
8954      \global\expandafter\let
8955        \csname\cf@encoding \string#1\expandafter\endcsname
8956        \csname ?\string#1\endcsname
8957    \fi
8958    \csname\cf@encoding\string#1%
8959      \expandafter\endcsname
8960  \else
8961    \noexpand#1%
8962  \fi
8963 }
8964 \def\@changed@x@err#1{%
8965    \errhelp{Your command will be ignored, type <return> to proceed}%
8966    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8967 \def\DeclareTextCommandDefault#1{%
8968    \DeclareTextCommand#1?%
8969 }
8970 \def\ProvideTextCommandDefault#1{%
8971    \ProvideTextCommand#1?%
8972 }
8973 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8974 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8975 \def\DeclareTextAccent#1#2#3{%
8976  \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8977 }
8978 \def\DeclareTextCompositeCommand#1#2#3#4{%
8979    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8980    \edef\reserved@b{\string##1}%
8981    \edef\reserved@c{%
8982     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8983    \ifx\reserved@b\reserved@c
8984      \expandafter\expandafter\expandafter\ifx
8985        \expandafter\@car\reserved@a\relax\relax\@nil
8986        \@text@composite
8987      \else
8988        \edef\reserved@b##1{%
8989          \def\expandafter\noexpand
8990            \csname#2\string#1\endcsname####1{%
8991            \noexpand\@text@composite
8992              \expandafter\noexpand\csname#2\string#1\endcsname
8993              ####1\noexpand\@empty\noexpand\@text@composite
8994              {##1}%
8995        }%
8996      }%
8997      \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8998    \fi
8999    \expandafter\def\csname\expandafter\string\csname
9000        #2\endcsname\string#1-\string#3\endcsname{#4}
9001  \else
9002    \errhelp{Your command will be ignored, type <return> to proceed}%
9003    \errmessage{\string\DeclareTextCompositeCommand\space used on
9004        inappropriate command \protect#1}
9005  \fi
9006 }
9007 \def\@text@composite#1#2#3\@text@composite{%
9008    \expandafter\@text@composite@x
9009      \csname\string#1-\string#2\endcsname
9010 }
9011 \def\@text@composite@x#1#2{%
```

```
9012    \ifx#1\relax
9013        #2%
9014    \else
9015        #1%
9016    \fi
9017 }
9018 %
9019 \def\@strip@args#1:#2-#3\@strip@args{#2}
9020 \def\DeclareTextComposite#1#2#3#4{%
9021    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9022    \bgroup
9023        \lccode`\@=#4%
9024        \lowercase{%
9025    \egroup
9026        \reserved@a @%
9027    }%
9028 }
9029 %
9030 \def\UseTextSymbol#1#2{#2}
9031 \def\UseTextAccent#1#2#3{}
9032 \def\@use@text@encoding#1{}
9033 \def\DeclareTextSymbolDefault#1#2{%
9034    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9035 }
9036 \def\DeclareTextAccentDefault#1#2{%
9037    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9038 }
9039 \def\cf@encoding{OT1}
```

Currently we only use the LaTeX 2_ε method for accents for those that are known to be made active in *some* language definition file.

```
9040 \DeclareTextAccent{\"}{OT1}{127}
9041 \DeclareTextAccent{\'}{OT1}{19}
9042 \DeclareTextAccent{\^}{OT1}{94}
9043 \DeclareTextAccent{\`}{OT1}{18}
9044 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
9045 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9046 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
9047 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
9048 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
9049 \DeclareTextSymbol{\i}{OT1}{16}
9050 \DeclareTextSymbol{\ss}{OT1}{25}
```

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

```
9051 \ifx\scriptsize\@undefined
9052   \let\scriptsize\sevenrm
9053 \fi
```

And a few more "dummy" definitions.

```
9054 \def\languagename{english}%
9055 \let\bbl@opt@shorthands\@nnil
9056 \def\bbl@ifshorthand#1#2#3{#2}%
9057 \let\bbl@language@opts\@empty
9058 \let\bbl@ensureinfo\@gobble
9059 \let\bbl@provide@locale\relax
9060 \ifx\babeloptionstrings\@undefined
9061   \let\bbl@opt@strings\@nnil
9062 \else
9063   \let\bbl@opt@strings\babeloptionstrings
9064 \fi
9065 \def\BabelStringsDefault{generic}
```

179

```
9066 \def\bbl@tempa{normal}
9067 \ifx\babeloptionmath\bbl@tempa
9068   \def\bbl@mathnormal{\noexpand\textormath}
9069 \fi
9070 \def\AfterBabelLanguage#1#2{}
9071 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
9072 \let\bbl@afterlang\relax
9073 \def\bbl@opt@safe{BR}
9074 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
9075 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
9076 \expandafter\newif\csname ifbbl@single\endcsname
9077 \chardef\bbl@bidimode\z@
9078 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
9079 ⟨∗plain⟩
9080 \input babel.def
9081 ⟨/plain⟩
```

15. Acknowledgements

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

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

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 LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

[3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.

[4] Donald E. Knuth, *The TeXbook*, Addison-Wesley, 1986.

[5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.

[6] Leslie Lamport, *LaTeX, A document preparation System*, Addison-Wesley, 1986.

[7] Leslie Lamport, in: TeXhax Digest, Volume 89, #13, 17 February 1989.

[8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.

[9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018

[10] Hubert Partl, *German TeX*, *TUGboat* 9 (1988) #1, p. 70–72.

[11] Joachim Schrod, *International LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

[12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using LaTeX*, Springer, 2002, p. 301–373.

[13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).