

Babel

Code

Version 26.14
2026/10/15

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

LuaT_EX

pdfT_EX

XeT_EX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	A few core definitions	8
3.2	$\LaTeX$: babel.sty (start)	8
3.3	base	10
3.4	key=value options and other general option	10
3.5	Post-process some options	12
3.6	Plain: babel.def (start)	13
4	babel.sty and babel.def (common)	14
4.1	Selecting the language	16
4.2	Errors	24
4.3	More on selection	25
4.4	Short tags	26
4.5	Compatibility with language.def	26
4.6	Hooks	27
4.7	Setting up language files	28
4.8	Shorthands	30
4.9	Language attributes	39
4.10	Support for saving and redefining macros	40
4.11	French spacing	42
4.12	Hyphens	42
4.13	Multiencoding strings	44
4.14	Tailor captions	49
4.15	Making glyphs available	50
4.15.1	Quotation marks	50
4.15.2	Letters	52
4.15.3	Shorthands for quotation marks	52
4.15.4	Umlauts and tremas	53
4.16	Layout	55
4.17	Load engine specific macros	55
4.18	Creating and modifying languages	55
4.19	Main loop in ‘provide’	63
4.20	Processing keys in ini	68
4.21	French spacing (again)	74
4.22	Handle language system	75
4.23	Numerals	76
4.24	Casing	78
4.25	Getting info	78
4.26	BCP 47 related commands	79
5	Adjusting the Babel behavior	80
5.1	Cross referencing macros	83
5.2	Layout	85
5.3	Marks	87
5.4	Other packages	88
5.4.1	ifthen	88
5.4.2	varioref	88
5.4.3	hhline	89
5.5	Encoding and fonts	90
5.6	Basic bidi support	91
5.7	Local Language Configuration	94
5.8	Language options	95

6	Tagging	99
7	The kernel of Babel	100
8	Error messages	100
9	Loading hyphenation patterns	104
10	luatex + xetex: common stuff	108
11	Hooks for XeTeX and LuaTeX	112
11.1	XeTeX	112
11.2	Support for interchar	113
11.3	Layout	115
11.4	8-bit TeX	117
11.5	LuaTeX	117
11.6	Southeast Asian scripts	124
11.7	CJK line breaking	126
11.8	Arabic justification	128
11.9	Common stuff	132
11.10	Automatic fonts and ids switching	133
11.11	Bidi	139
11.12	Layout	142
11.13	Lua: transforms	151
11.14	Lua: Auto bidi with basic and basic-r	162
12	Data for CJK	175
13	The ‘nil’ language	175
14	Calendars	176
14.1	Islamic	176
14.2	Hebrew	178
14.3	Persian	182
14.4	Coptic and Ethiopic	183
14.5	Julian	183
14.6	Buddhist	184
15	Support for Plain TeX (plain.def)	185
15.1	Not renaming hyphen.tex	185
15.2	Emulating some L ^A T _E X features	186
15.3	General tools	186
15.4	Encoding related macros	190
16	Acknowledgements	193

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

1. Identification and loading of required files

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files.

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

2. locale directory

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

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

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

3. Tools

```
1 <<version=26.14>>
2 <<date=2026/10/15>>
```

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

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

```

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

```

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

```

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

```

\bbl@afterelse

\bbl@afterfi Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an \if-statement¹. These macros will break if another \if... \fi statement appears in one of the arguments and it is not enclosed in braces.

```

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

```

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

```

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

```

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

```

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

```

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

\bbl@ifunset 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 `\ifcurname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcurname` being implicitly set to `\relax` by the `\curname` test.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

196 \def\bb@array@end{\@empty}
197 \def\bb@array@set#1#2#3{%
198   \def\bb@array@aux##1\bb@elt#2\@##2##3\bb@array@aux##4{%
199     \ifx\bb@noelt##2%
200       \def#1{##1\bb@elt#2\@{##4}}%
201     \else
202       \def\bb@array@aux####1\bb@elt#2\@####2####3\bb@array@aux####4{%
203         \def#1{####1\bb@elt#2\@{####4}####3}}%
204       \expandafter\bb@array@aux#1\bb@array@aux{#3}%
205     \fi}%
206   \expandafter\bb@array@aux#1\bb@elt#2\@{\bb@noelt\bb@array@aux{#3}}
207 \def\bb@array@get#1#2#3{%
208   \def\bb@array@aux##1\bb@elt#2\@##2##3\bb@array@aux{%
209     \ifx\bb@array@end##2\@empty\let#3\relax\else\def#3{##2}\fi}%

```

```
210 \expandafter\bbbl@array@aux#1\bbbl@elt#2\@@\bbbl@array@end\bbbl@array@aux}
```

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

```
211 \def\bbbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
212 \toks@{\expandafter\expandafter\expandafter{%
213 \cscname extras\language\endcsname}%
214 \bbbl@exp{\in@{#1}{\the\toks@}}}%
215 \ifin@else
216 \@temptokena{#2}%
217 \edef\bbbl@tempc{\the\@temptokena\the\toks@}%
218 \toks@\expandafter{\bbbl@tempc#3}%
219 \expandafter\edef\cscname extras\language\endcsname{\the\toks@}%
220 \fi}
221 <</Basic macros>>
```

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

```
222 <<*Make sure ProvidesFile is defined>> ≡
223 \ifx\ProvidesFile\@undefined
224 \def\ProvidesFile#1[#2 #3 #4]{%
225 \wlog{File: #1 #4 #3 <#2>}%
226 \let\ProvidesFile\@undefined}
227 \fi
228 <</Make sure ProvidesFile is defined>>
```

3.1. A few core definitions

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

```
229 <<*Define core switching macros>> ≡
230 \ifx\language\@undefined
231 \cscname newcount\endcsname\language
232 \fi
233 <</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.

```
234 <<*Define core switching macros>> ≡
235 \countdef\last@language=19
236 \def\addlanguage{\cscname newlanguage\endcsname}
237 <</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.

```
238 <*package>
239 \NeedsTeXFormat{LaTeX2e}
240 \ProvidesPackage{babel}%
241 [<@date@> v<@version@>
242 The multilingual framework for LuaLaTeX, pdfLaTeX and XeLaTeX]
```

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

```

243 \ifpackagewith{babel}{debug}
244 {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
245 \let\bbl@debug\@firstofone
246 \ifx\directlua\@undefined\else
247 \directlua{
248 Babel = Babel or {}
249 Babel.debug = true }%
250 \input{babel-debug.tex}%
251 \fi}
252 {\providecommand\bbl@trace[1]{}%
253 \let\bbl@debug\@gobble
254 \ifx\directlua\@undefined\else
255 \directlua{
256 Babel = Babel or {}
257 Babel.debug = false }%
258 \fi}

```

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

```

259 \def\bbl@error#1{% Implicit #2#3#4
260 \begingroup
261 \catcode`\=0 \catcode`\==12 \catcode`\`=12
262 \input errbabel.def
263 \endgroup
264 \bbl@error{#1}}
265 \def\bbl@warning#1{%
266 \begingroup
267 \def\{\MessageBreak}%
268 \PackageWarning{babel}{#1}%
269 \endgroup}
270 \def\bbl@infowarn#1{%
271 \begingroup
272 \def\{\MessageBreak}%
273 \PackageNote{babel}{#1}%
274 \endgroup}
275 \def\bbl@info#1{%
276 \begingroup
277 \def\{\MessageBreak}%
278 \PackageInfo{babel}{#1}%
279 \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.

```

280 <@Basic macros>
281 \ifpackagewith{babel}{silent}
282 {\let\bbl@info\@gobble
283 \let\bbl@infowarn\@gobble
284 \let\bbl@warning\@gobble}
285 {}
286 %
287 \def\AfterBabelLanguage#1{%
288 \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.

```

289 \ifx\bbl@languages\@undefined\else
290 \begingroup
291 \catcode`\^^I=12
292 \@ifpackagewith{babel}{showlanguages}{%
293 \begingroup
294 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%

```

```

295      \wlog{<*languages>}%
296      \bbl@languages
297      \wlog{</languages>}%
298      \endgroup{}}
299  \endgroup
300  \def\bbl@elt#1#2#3#4{%
301    \ifnum#2=\z@
302      \gdef\bbl@nulllanguage{#1}%
303      \def\bbl@elt##1##2##3##4{%
304        \fi}%
305    \bbl@languages
306  \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 $\TeX$ forgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.

Now the base option. With it we can define (and load, with `luatex`) hyphenation patterns, even if we are not interested in the rest of `babel`.

```

307 \bbl@trace{Defining option 'base'}
308 \ifpackagewith{babel}{base}{%
309   \let\bbl@onlyswitch\@empty
310   \let\bbl@provide@locale\relax
311   \input babel.def
312   \let\bbl@onlyswitch\@undefined
313   \ifx\directlua\@undefined
314     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
315   \else
316     \input luababel.def
317     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
318   \fi
319   \DeclareOption{base}{}%
320   \DeclareOption{showlanguages}{}%
321   \ProcessOptions
322   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
323   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
324   \global\let\@ifl@ter@\@ifl@ter
325   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@}%
326   \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`.

```

327 \bbl@trace{key=value and another general options}
328 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
329 \def\bbl@tempb#1.#2{% Removes trailing dot
330   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
331 \def\bbl@tempe#1=#2\@{
332   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
333 \def\bbl@tempd#1.#2\@nnil{%
334   \ifx\@empty#2%
335     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
336   \else
337     \in@{,provide=}{, #1}%
338     \ifin@
339       \edef\bbl@tempc{%
340         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
341     \else
342       \in@{$modifiers$}{$#1$}%

```

```

343 \ifin@
344 \bbl@tempe#2\@@
345 \else
346 \in@{=}{#1}%
347 \ifin@
348 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
349 \else
350 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
351 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
352 \fi
353 \fi
354 \fi
355 \fi}
356 \let\bbl@tempc\@empty
357 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
358 \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.

```

359 \DeclareOption{KeepShorthandsActive}{}
360 \DeclareOption{activeacute}{}
361 \DeclareOption{activegrave}{}
362 \DeclareOption{debug}{}
363 \DeclareOption{debug-german}{} % Temporary
364 \DeclareOption{noconfigs}{}
365 \DeclareOption{showlanguages}{}
366 \DeclareOption{silent}{}
367 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
368 \chardef\bbl@iniflag\z@
369 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
370 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
371 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
372 \chardef\bbl@ldfflag\z@
373 \DeclareOption{provide=!}{\chardef\bbl@ldfflag\@ne} % main = 1
374 \DeclareOption{provide+=!}{\chardef\bbl@ldfflag\tw@} % second = 2
375 \DeclareOption{provide*=!}{\chardef\bbl@ldfflag\thr@@} % second + main
376 \DeclareOption{licr=extended}{}
377 \DeclareOption{licr=unextended}{}
378 \let\bbl@ifmetadata\@gobble
379 \DeclareOption{babel-metadata=on}{%
380 \IfDocumentMetadataT{\AtEndOfPackage{\input{tagbabel.def}}}}
381 \DeclareOption{babel-metadata=tagging}{%
382 \IfDocumentMetadataT{\AtEndOfPackage{\input{tagbabel.def}}}}
383 % Don't use. Experimental.
384 \newif\ifbbl@single
385 \DeclareOption{selectors=off}{\bbl@singletrue}
386 <@More package options@>

```

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

```

387 \let\bbl@opt@shorthands\@nnil
388 \let\bbl@opt@config\@nnil
389 \let\bbl@opt@main\@nnil
390 \let\bbl@opt@headfoot\@nnil
391 \let\bbl@opt@layout\@nnil
392 \let\bbl@opt@provide\@nnil

```

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

```

393 \def\bbl@tempa#1=#2\bbl@tempa{%
394 \bbl@csarg\ifx{opt@#1}\@nnil
395 \bbl@csarg\edef{opt@#1}{#2}%

```

```

396 \else
397   \bbl@error{bad-package-option}{#1}{#2}{}%
398 \fi}

```

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

```

399 \let\bbl@language@opts@empty
400 \DeclareOption*{%
401   \bbl@xin@{\string=}{\CurrentOption}%
402   \ifin@
403     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
404   \else
405     \bbl@add@list\bbl@language@opts{\CurrentOption}%
406   \fi}

```

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

```

407 \ProcessOptions*

```

3.5. Post-process some options

```

408 \ifx\bbl@opt@provide@nnil
409   \let\bbl@opt@provide@empty %%% MOVE above
410 \else
411   \chardef\bbl@iniflag@ne
412   \bbl@expf{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
413     \in@{,provide,}{, #1,}%
414     \ifin@
415       \def\bbl@opt@provide{#2}%
416     \fi}
417 \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=...`

```

418 \bbl@trace{Conditional loading of shorthands}
419 \def\bbl@sh@string#1{%
420   \ifx#1@empty\else
421     \ifx#1t\string~%
422     \else\ifx#1c\string,%
423     \else\string#1%
424   \fi\fi
425   \expandafter\bbl@sh@string
426 \fi}
427 \ifx\bbl@opt@shorthands@nnil
428   \def\bbl@ifshorthand#1#2#3{#2}%
429 \else\ifx\bbl@opt@shorthands@empty
430   \def\bbl@ifshorthand#1#2#3{#3}%
431 \else

```

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

```

432 \def\bbl@ifshorthand#1{%
433   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
434   \ifin@
435     \expandafter\@firstoftwo
436   \else
437     \expandafter\@secondoftwo
438   \fi}

```

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

```

439 \edef\bbl@opt@shorthands{%
440   \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.

```
441 \bbl@ifshorthand{'}%
442 {\PassOptionsToPackage{activeacute}{babel}}{}
443 \bbl@ifshorthand{'}%
444 {\PassOptionsToPackage{activegrave}{babel}}{}
445 \fi\fi
```

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

```
446 \ifx\bbl@opt@headfoot\@nnil\else
447 \g@addto@macro\@resetactivechars{%
448 \set@typeset@protect
449 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
450 \let\protect\noexpand}
451 \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`.

```
452 \ifx\bbl@opt@safe\@undefined
453 \def\bbl@opt@safe{BR}
454 % \let\bbl@opt@safe\@empty % Pending of \cite
455 \fi
```

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

```
456 \bbl@trace{Defining IfBabelLayout}
457 \ifx\bbl@opt@layout\@nnil
458 \newcommand\IfBabelLayout[3]{#3}%
459 \else
460 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
461 \in@{, layout,}{, #1,}%
462 \ifin@
463 \def\bbl@opt@layout{#2}%
464 \bbl@replace\bbl@opt@layout{ }{.}%
465 \fi}
466 \newcommand\IfBabelLayout[1]{%
467 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
468 \ifin@
469 \expandafter\@firstoftwo
470 \else
471 \expandafter\@secondoftwo
472 \fi}
473 \fi
474 (/package)
```

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

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

First, exit immediately if previously loaded.

```
475 (*core)
476 \ifx\ldf@quit\@undefined\else
477 \endinput\fi % Same line!
478 <@Make sure ProvidesFile is defined@>
479 \ProvidesFile{babel.def}[<@date@> v<@version@> Babel common definitions]
480 \ifx\AtBeginDocument\@undefined
481 <@Emulate LaTeX@>
482 \fi
483 <@Basic macros@>
484 (/core)
```

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

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

```

485 (*package|core)
486 \def\bbl@version{<@version>}
487 \def\bbl@date{<@date>}
488 <@Define core switching macros>

```

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

```

489 \def\addialect#1#2{%
490   \global\chardef#1#2\relax
491   \bbl@usehooks{addialect}{#1}{#2}}%
492 \begingroup
493   \count@#1\relax
494   \def\bbl@elt##1##2##3##4{%
495     \ifnum\count@=#2\relax
496       \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
497       \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
498         set to \expandafter\string\csname l@##1\endcsname\%
499         (\string\language\the\count@). Reported}%
500       \def\bbl@elt####1####2####3####4{%}
501       \fi}%
502   \bbl@cs{languages}%
503 \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.

```

504 \def\bbl@fixname#1{%
505   \begingroup
506   \def\bbl@tempe{l@}%
507   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
508   \bbl@tempd
509   {\lowercase\expandafter{\bbl@tempd}%
510    {\uppercase\expandafter{\bbl@tempd}%
511     \@empty
512     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
513      \uppercase\expandafter{\bbl@tempd}}}%
514    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
515     \lowercase\expandafter{\bbl@tempd}}}%
516   \@empty
517   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
518   \bbl@tempd
519   \bbl@exp{\bbl@usehooks{languagename}{\language}{#1}}%
520 \def\bbl@iflanguage#1{%
521   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

We first need a couple of macros for a simple BCP 47 look up. It also makes sure, casing is the usual one, so that sr-latn-ba becomes fr-Latn-BA.

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

```

522 \def\bbl@cntchars#1{\bbl@cntchars@i#1876543210\@@}
523 \def\bbl@cntchars@i#1#2#3#4#5#6#7#8#9\@{\@car#9\@nil}%
524 \def\bbl@bcplookup#1{%
525   \let\bbl@templ\@empty % language
526   \let\bbl@temps\@empty % script

```

```

527 \let\bbl@tempr\@empty % regions
528 \let\bbl@tempv\@empty % variant
529 \edef\bbl@tempa{#1}%
530 \bbl@replace\bbl@tempa{-}{,}%
531 \bbl@replace\bbl@tempa_{}{,}%
532 \bbl@foreach\bbl@tempa{%
533   \ifx\bbl@templ\@empty
534     \lowercase{\def\bbl@templ{##1}}%
535   \else\ifnum\bbl@cntchars{##1}=4
536     \bbl@xin@{\@car##1\@nil}{0123456789}%
537     \ifin@
538       \def\bbl@tempv{##1}% eg, 1901
539     \else
540       \uppercase{\edef\bbl@tempb{\@car##1\@nil}}%
541       \lowercase{\edef\bbl@tempb{\bbl@tempb\@gobble##1}}%
542     \fi
543   \else\ifnum\bbl@cntchars{##1}<4
544     \uppercase{\def\bbl@tempr{##1}}%
545   \else\ifnum\bbl@cntchars{##1}>4
546     \lowercase{\def\bbl@tempv{##1}}%
547   \fi\fi\fi\fi}%
548 \bbl@exp{\\bbl@bcpllookup@i{\bbl@templ}{\bbl@tempb}{\bbl@tempr}{\bbl@tempv}}
549 \def\bbl@bcpllookup@try#1{%
550   \ifx\bbl@bcp\relax
551     \IfFileExists{babel-#1\bbl@tempe.ini}{\edef\bbl@bcp{#1\bbl@tempe}}{}%
552   \fi}
553 \def\bbl@bcpllookup@i#1#2#3#4{%
554   \let\bbl@bcp\relax
555   \def\bbl@tempe{#4}%
556   \ifx\bbl@tempe\@empty\else
557     \edef\bbl@tempe{-#4}%
558   \fi
559   \edef\bbl@tempa{#2#3\bbl@tempe}% en
560   \ifx\bbl@tempa\@empty
561     \bbl@bcpllookup@try{#1}%
562   \else
563     \edef\bbl@tempa{#3\bbl@tempe}% en-Latn
564     \ifx\bbl@tempa\@empty
565       \bbl@bcpllookup@try{#1-#2}%
566       \bbl@bcpllookup@try{#1}%
567     \else
568       \edef\bbl@tempa{#2\bbl@tempe}% en-US, en-001
569       \ifx\bbl@tempa\@empty
570         \bbl@bcpllookup@try{#1-#3}%
571         \bbl@bcpllookup@try{#1}%
572       \else % en-Latn-US
573         \bbl@bcpllookup@try{#1-#2-#3}%
574         \bbl@bcpllookup@try{#1-#2}%
575         \bbl@bcpllookup@try{#1-#3}%
576         \bbl@bcpllookup@try{#1}%
577       \fi
578     \fi
579   \fi
580   \ifx\bbl@bcp\relax
581     \ifx\bbl@tempe\@empty\else\bbl@bcpllookup@i{#1}{#2}{#3}{#4}\fi%
582   \fi}
583 %
584 \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.

```
585 \def\iflanguage#1{%
586   \bbl@iflanguage{#1}{%
587     \ifnum\csname l@#1\endcsname=\language
588       \expandafter\@firstoftwo
589     \else
590       \expandafter\@secondoftwo
591     \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.

```
592 \let\bbl@select@type\z@
593 \edef\selectlanguage{%
594   \noexpand\protect
595   \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`.

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

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

```
597 \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.

```
598 \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:

```
599 \def\bbl@push@language{%
600   \ifx\language\@undefined\else
601     \ifx\currentgrouplevel\@undefined
602       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
603     \else
604       \ifnum\currentgrouplevel=\z@
605         \xdef\bbl@language@stack{\language+}%
606       \else
607         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
608       \fi
609     \fi
610   \fi}
```

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

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

```
611 \def\bbl@pop@lang#1+#2\@@{%
612   \edef\language{#1}%
613   \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 T_EX 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).

```
614 \let\bbl@ifrestoring\@secondoftwo
615 \def\bbl@pop@language{%
616   \expandafter\bbl@pop@lang\bbl@language@stack\@@
617   \let\bbl@ifrestoring\@firstoftwo
618   \expandafter\bbl@set@language\expandafter{\language}%
619   \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).

```
620 \chardef\localeid\z@
621 \gdef\bbl@id@last{0}    % No real need for a new counter
622 \def\bbl@id@assign{%
623   \bbl@ifunset\bbl@id@\language}%
624   {\count@\bbl@id@last\relax
625    \advance\count@\@ne
626    \global\bbl@csarg\chardef{id@\language}\count@
627    \xdef\bbl@id@last{\the\count@}%
628    \ifcase\bbl@engine\or
629      \directlua{
630        Babel.locale_props[\bbl@id@last] = {}
631        Babel.locale_props[\bbl@id@last].name = '\language'
632        Babel.locale_props[\bbl@id@last].vars = {}
633      }%
634    \fi}%
635  }%
636  \chardef\localeid\bbl@c{l{id@}}
```

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

```
637 \let\bbl@select@opts\@empty
638 \expandafter\def\csname selectlanguage \endcsname{%
639   \@ifnextchar[\bbl@select@s{\bbl@select@s[]}}
640 \def\bbl@select@s[#1]#2{%
641   \bbl@forkv{#1}{\bbl@array@set\bbl@select@opts{##1}{##2}}%
642   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw\fi
643   \bbl@push@language
644   \aftergroup\bbl@pop@language
645   \bbl@set@language{#2}}
646 \let\endselectlanguage\relax
```

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

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

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

```

647 \def\BabelContentsFiles{toc,lof,lot}
648 \def\bbl@set@language#1{% from selectlanguage, pop@
649 % The old buggy way. Preserved for compatibility, but simplified
650 \edef\language{\expandafter\string#1\@empty}%
651 \select@language{\language}%
652 \bbl@array@get\bbl@select@opts{main}\bbl@tempa
653 \ifx\bbl@tempa\relax\else
654   \let\bbl@main@language\localename
655   \let\mainlocalename\localename
656 \fi
657 % write to aux files
658 \expandafter\ifx\csname date\language\endcsname\relax\else
659   \if@filesw
660     \bbl@array@get\bbl@select@opts{nofiles}\bbl@tempa
661     \ifx\bbl@tempa\relax
662       \ifx\babel@aux@gobbletwo\else % Set if single in the first, redundant
663         \bbl@savelastskip
664         \protected@write\@auxout{\}\string\babel@aux{\bbl@auxname}{}}%
665         \bbl@restorelastskip
666       \fi
667       \bbl@usehooks{write}{}%
668     \fi
669   \fi
670 \fi
671 \let\bbl@select@opts\@empty}
672 %
673 \let\bbl@restorelastskip\relax
674 \let\bbl@savelastskip\relax
675 %
676 \def\select@language#1{% from set@, babel@aux, babel@toc
677 \ifx\bbl@select@name\@empty
678   \def\bbl@select@name{select}%
679 \fi
680 % set hymap
681 \ifnum\bbl@hymapset=\@cclv\chardef\bbl@hymapset4\relax\fi
682 % set name (when coming from babel@aux)
683 \edef\language{#1}%
684 \bbl@fixname\language
685 % define \localename when coming from set@, with a trick
686 \ifx\scantokens\undefined
687   \def\localename{??}%
688 \else
689   \bbl@exp{\scantokens{\def\localename{\language}\noexpand}\relax}%
690 \fi
691 \bbl@provide@locale
692 \bbl@iflanguage\language{%
693   \let\bbl@select@type\z@
694   \expandafter\bbl@switch\expandafter{\language}}
695 \def\babel@aux#1#2{%
696   \select@language{#1}%
697   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
698     \writefile{##1}{\babel@toc{#1}{#2}\relax}}%
699 \def\babel@toc#1#2{%
700   \select@language{#1}}

```

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

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

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To

save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<language>` command at definition time by expanding the `\csname` primitive.

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

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

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

```

701 \newif\ifbbl@usedategroup
702 \let\bbl@savextras\@empty
703 \def\bbl@switch#1{% from select@, foreign@
704   % restore
705   \originalTeX
706   \expandafter\def\expandafter\originalTeX\expandafter{%
707     \csname noextras#1\endcsname
708     \let\originalTeX\@empty
709     \babel@beginsave}%
710   \bbl@usehooks{afterreset}{}%
711   \languageshorthands{none}%
712   % set the locale id
713   \bbl@id@assign
714   % switch captions, date
715   \bbl@bsphack
716   \ifcase\bbl@select@type
717     \csname captions#1\endcsname\relax
718     \csname date#1\endcsname\relax
719   \else
720     \bbl@array@get\bbl@select@opts{captions}\bbl@tempa
721     \ifx\bbl@tempa\relax\else
722       \csname captions#1\endcsname\relax
723     \fi
724     \bbl@array@get\bbl@select@opts{date}\bbl@tempa
725     \ifx\bbl@tempa\relax\else % if \foreign... within \<language>date
726       \csname date#1\endcsname\relax
727     \fi
728   \fi
729   \bbl@esphack
730   % switch extras
731   \csname bbl@preextras@#1\endcsname
732   \bbl@usehooks{beforeextras}{}%
733   \csname extras#1\endcsname\relax
734   \bbl@usehooks{afterextras}{}%
735   % > babel-ensure
736   % > babel-sh-<short>
737   % > babel-bidi
738   % > babel-fontspec
739   \let\bbl@savextras\@empty
740   % hyphenation - case mapping
741   \ifcase\bbl@opt@hyphenmap\or
742     \def\BabelLower##1##2{\lccode##1=##2\relax}%
743     \ifnum\bbl@hymapsel>4\else
744       \csname\language @bbl@hyphenmap\endcsname
745     \fi
746     \chardef\bbl@opt@hyphenmap\z@
747   \else
748     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
749       \csname\language @bbl@hyphenmap\endcsname
750     \fi
751   \fi
752   \let\bbl@hymapsel\@cclv
753   % hyphenation - select rules

```

```

754 \ifnum\csname l@\language\endcsname=\l@unhyphenated
755 \edef\bbl@tempa{u}%
756 \else
757 \edef\bbl@tempa{\bbl@cl{lbrk}}%
758 \fi
759 % linebreaking - handle u, e, k (v in the future)
760 \bbl@xin@{/u}{/\bbl@tempa}%
761 \ifin@else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
762 \ifin@else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
763 \ifin@else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (e.g., Tibetan)
764 \ifin@else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
765 % hyphenation - save mins
766 \babel@savevariable\lefthyphenmin
767 \babel@savevariable\righthyphenmin
768 \ifnum\bbl@engine=\@ne
769 \babel@savevariable\hyphenationmin
770 \fi
771 \ifin@
772 % unhyphenated/kashida/elongated/padding = allow stretching
773 \language\l@unhyphenated
774 \babel@savevariable\emergencystretch
775 \emergencystretch\maxdimen
776 \babel@savevariable\hbadness
777 \hbadness\@M
778 \else
779 % other = select patterns
780 \bbl@patterns{#1}%
781 \fi
782 % hyphenation - set mins
783 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
784 \set@hyphenmins\tw@\thr@@\relax
785 \@nameuse{\bbl@hyphenmins@}%
786 \else
787 \expandafter\expandafter\expandafter\set@hyphenmins
788 \csname #1hyphenmins\endcsname\relax
789 \fi
790 \@nameuse{\bbl@hyphenmins@}%
791 \@nameuse{\bbl@hyphenmins@\language}%
792 \@nameuse{\bbl@hyphenatmin@}%
793 \@nameuse{\bbl@hyphenatmin@\language}%
794 \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.

```

795 \edef\otherlanguage{%
796 \noexpand\protect
797 \expandafter\noexpand\csname otherlanguage \endcsname}
798 \expandafter\def\csname otherlanguage \endcsname{%
799 \@ifstar{\@nameuse{otherlanguage*}}{\bbl@otherlanguage}
800 \def\bbl@otherlanguage#1{%
801 \def\bbl@selectorname{other}%
802 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
803 \csname selectlanguage \endcsname{#1}%
804 \ignorespaces}

```

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

```

805 \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.`

```

806 \expandafter\def\csname otherlanguage*\endcsname{%
807   \ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
808 \def\bbl@otherlanguage@s[#1]#2{%
809   \def\bbl@selectorname{other*}%
810   \ifnum\bbl@hymapset=\ccclv\chardef\bbl@hymapset4\relax\fi
811   \bbl@forkv{#1}{\bbl@array@set\bbl@select@opts{##1}{##2}}%
812   \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”.

```

813 \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.

```

814 \providecommand\bbl@beforeforeign{}
815 \edef\foreignlanguage{%
816   \noexpand\protect
817   \expandafter\noexpand\csname foreignlanguage \endcsname}
818 \expandafter\def\csname foreignlanguage \endcsname{%
819   \@ifstar\bbl@foreign@s\bbl@foreign@x}
820 \providecommand\bbl@foreign@x[3][[]]{%
821   \begingroup
822     \def\bbl@selectorname{foreign}%
823     \bbl@forkv{#1}{\bbl@array@set\bbl@select@opts{##1}{##2}}%
824     \let\BabelText\@firstofone
825     \bbl@beforeforeign
826     \foreign@language{#2}%
827     \bbl@ifmetadata{\UseTaggingSocket{babel/foreign/begin}}%
828     \bbl@usehooks{foreign}{}%
829     \BabelText{#3}% Now in horizontal mode!
830     \bbl@ifmetadata{\UseTaggingSocket{babel/foreign/end}}%
831   \endgroup}
832 \def\bbl@foreign@s#1#2{%
833   \begingroup
834     {\par}%
835     \def\bbl@selectorname{foreign*}%
836     \let\bbl@select@opts\@empty
837     \let\BabelText\@firstofone
838     \foreign@language{#1}%
839     \bbl@usehooks{foreign*}{}%
840     \bbl@dirparastext
841     \BabelText{#2}% Still in vertical mode!
842   {\par}%

```

```

843 \endgroup}
844 \providecommand\BabelWrapText[1]{%
845 \def\bbl@tempa{\def\BabelText###1}%
846 \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.

```

847 \def\foreign@language#1{%
848 % set name
849 \edef\language#1%
850 \ifbbl@usedategroup
851 \bbl@array@set\bbl@select@opts{date}{\@empty}%
852 \bbl@usedategroupfalse
853 \fi
854 \bbl@fixname\language
855 \let\localename\language
856 \bbl@provide@locale
857 \bbl@iflanguage\language{%
858 \let\bbl@select@type\@ne
859 \expandafter\bbl@switch\expandafter{\language}}

```

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

```

860 \def\IfBabelSelectorTF#1{%
861 \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
862 \ifin@
863 \expandafter\@firstoftwo
864 \else
865 \expandafter\@secondoftwo
866 \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.

```

867 \let\bbl@hyphlist\@empty
868 \let\bbl@hyphenation@\relax
869 \let\bbl@pttnlist\@empty
870 \let\bbl@patterns@\relax
871 \let\bbl@hymapsel=\@cclv
872 \def\bbl@patterns#1{%
873 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
874 \csname l@#1\endcsname
875 \edef\bbl@tempa{#1}%
876 \else
877 \csname l@#1:\f@encoding\endcsname
878 \edef\bbl@tempa{#1:\f@encoding}%
879 \fi
880 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
881 % > luatex
882 \ifundefined{bbl@hyphenation@}{% Can be \relax!
883 \begingroup
884 \bbl@xin@{\number\language,}{,\bbl@hyphlist}%
885 \ifin@else
886 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
887 \hyphenation{%
888 \bbl@hyphenation@
889 \ifundefined{bbl@hyphenation@#1}%

```

```

890         \@empty
891         {\space\csname bbl@hyphenation@#1\endcsname}}%
892         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
893         \fi
894     \endgroup}}

```

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

```

895 \def\hyphenrules#1{%
896     \edef\bbl@tempf{#1}%
897     \bbl@fixname\bbl@tempf
898     \bbl@iflanguage\bbl@tempf{%
899         \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
900         \ifx\languageshorthands\@undefined\else
901             \languageshorthands{none}%
902         \fi
903         \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
904             \set@hyphenmins\tw@thr@@\relax
905         \else
906             \expandafter\expandafter\expandafter\set@hyphenmins
907             \csname\bbl@tempf hyphenmins\endcsname\relax
908         \fi}}
909 \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.

```

910 \def\providehyphenmins#1#2{%
911     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
912         \@namedef{#1hyphenmins}{#2}%
913     \fi}

```

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

```

914 \def\set@hyphenmins#1#2{%
915     \lefthyphenmin#1\relax
916     \righthyphenmin#2\relax}

```

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

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

```

917 \ifx\ProvidesFile\@undefined
918     \def\ProvidesLanguage#1[#2 #3 #4]{%
919         \wlog{Language: #1 #4 #3 <#2>}%
920     }
921 \else
922     \def\ProvidesLanguage#1{%
923         \begingroup
924             \catcode`\ 10 %
925             \@makeother\/%
926             \@ifnextchar[%]
927                 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
928     \def\@provideslanguage#1[#2]{%
929         \wlog{Language: #1 #2}%
930         \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
931     \endgroup}
932 \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`.

```
933 \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.

```
934 \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’:

```
935 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}{}
936 \let\uselocale\setlocale
937 \let\locale\setlocale
938 \let\selectlocale\setlocale
939 \let\textlocale\setlocale
940 \let\textlanguage\setlocale
941 \let\languagegettext\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 $\LaTeX 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

```
942 \edef\bbl@nulllanguage{\string\language=0}
943 \def\bbl@nocaption#1#2{\NoCaseChange{\protect\bbl@nocaption@i{#1}{#2}}}
944 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
945   \global\@namedef{#2}{\textbf{?#1?}}}%
946   \@nameuse{#2}%
947   \edef\bbl@tempa{#1}%
948   \bbl@sreplace\bbl@tempa{name}}}%
949   \bbl@warning{%
950     \@backslashchar#1 not set for '\language'. Please,\\%
951     define it after the language has been loaded\\%
952     (typically in the preamble) with:\\%
953     \string\setlocalecaption{\language}\bbl@tempa{.}\\%
954     Feel free to contribute on github.com/latex3/babel.\\%
955     Reported}}
956 \def\bbl@tentative{\protect\bbl@tentative@i}
957 \def\bbl@tentative@i#1{%
958   \bbl@warning{%
959     Some functions for '#1' are tentative.\\%
960     They might not work as expected and their behavior\\%
961     could change in the future.\\%
962     Reported}}
963 \def\@nolanerr#1{\bbl@error{undefined-language}{#1}}{}{}
964 \def\@nopatterns#1{%
965   \bbl@warning
966     {No hyphenation patterns were preloaded for\\%
967     the language '#1' into the format.\\%
968     Please, configure your TeX system to add them and\\%
969     rebuild the format. Now I will use the patterns\\%
970     preloaded for \bbl@nulllanguage\space instead}}
971 \let\bbl@usehooks@gobbletwo
```

Here ended the now discarded switch.def.
 Here also (currently) ends the base option.
 972 \ifx\bbl@onlyswitch\@empty\endinput\fi

4.3. More on selection

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

The macro `\bbl@e@<language>` contains `\bbl@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

973 \bbl@trace{Defining babelensure}
974 \newcommand\babelensure[2][]{%
975   \AddBabelHook{babel-ensure}{afterextras}{%
976     \ifcase\bbl@select@type
977       \bbl@c{l}{e}%
978     \fi}%
979   \begingroup
980     \let\bbl@ens@include\@empty
981     \let\bbl@ens@exclude\@empty
982     \def\bbl@ens@fontenc{\relax}%
983     \def\bbl@tempb##1{%
984       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
985     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
986     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
987     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
988     \def\bbl@tempc{\bbl@ensure}%
989     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
990       \expandafter{\bbl@ens@include}}%
991     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
992       \expandafter{\bbl@ens@exclude}}%
993     \toks@\expandafter{\bbl@tempc}%
994     \bbl@exp{%
995   \endgroup
996   \def<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
997 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
998 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
999   \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1000     \edef##1{\noexpand\bbl@nocaption
1001       {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
1002   \fi
1003   \ifx##1\@empty\else
1004     \in@{##1}{#2}%
1005     \ifin@ \else
1006       \let\bbl@tempc\language\name
1007       \bbl@replace\bbl@tempc{-}{@}%
1008       \bbl@ifunset{\bbl@ensure@\bbl@tempc}%
1009       {\bbl@exp{%
1010         \\DeclareRobustCommand\<\bbl@ensure@\bbl@tempc>[1]{%
1011           \\foreignlanguage{\language\name}%
1012           {\ifx\relax#3\else
1013             \\fontencoding{#3}\\selectfont
1014           \fi
1015           #####1}}}%
1016       {}%
1017       \toks@\expandafter{##1}%
1018       \edef##1{%
1019         \bbl@csarg\noexpand{ensure@\bbl@tempc}%

```

```

1020         {\the\toks@}}}%
1021     \fi
1022     \expandafter\babel@tempb
1023 \fi}%
1024 \expandafter\babel@tempb\babel@captionslist\today\@empty
1025 \def\babel@tempa##1{% elt for include list
1026     \ifx##1\@empty\else
1027         \let\babel@tempc\language\language
1028         \babel@replace\babel@tempc{-}{@}%
1029         \babel@csarg\in@{ensure@\babel@tempc\expandafter}\expandafter{##1}%
1030         \ifin@\else
1031             \babel@tempb##1\@empty
1032         \fi
1033         \expandafter\babel@tempa
1034     \fi}%
1035 \babel@tempa#1\@empty}
1036 \def\babel@captionslist{%
1037     \prefacename\refname\abstractname\bibname\chaptername\appendixname
1038     \contentsname\listfigurename\listtablename\indexname\figurename
1039     \tablename\partname\enclname\ccname\headtoname\pagename\seename
1040     \alsoname\proofname\glossaryname}

```

4.4. Short tags

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

```

1041 \babel@trace{Short tags}
1042 \newcommand\babeltags[1]{%
1043     \edef\babel@tempa{\zap@space#1 \@empty}%
1044     \def\babel@tempb##1=##2\@{
1045         \edef\babel@tempc{
1046             \noexpand\newcommand
1047             \expandafter\noexpand\csname ##1\endcsname{
1048                 \noexpand\protect
1049                 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1050             \noexpand\newcommand
1051             \expandafter\noexpand\csname text##1\endcsname{
1052                 \noexpand\foreignlanguage{##2}}}
1053     \babel@tempc}%
1054 \babel@for\babel@tempa\babel@tempa{
1055     \expandafter\babel@tempb\babel@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.

```

1056 \babel@trace{Compatibility with language.def}
1057 \ifx\directlua\@undefined\else
1058     \ifx\babel@luapatterns\@undefined
1059         \input luababel.def
1060     \fi
1061 \fi
1062 \ifx\babel@languages\@undefined
1063     \ifx\directlua\@undefined
1064         \openin0 = language.def
1065         \ifeof0
1066             \closein0
1067             \message{I couldn't find the file language.def}
1068         \else
1069             \closein0
1070             \begingroup
1071             \def\addlanguage#1#2#3#4#5{

```

```

1072      \expandafter\ifx\csname lang@#1\endcsname\relax\else
1073      \global\expandafter\let\csname l@#1\expandafter\endcsname
1074      \csname lang@#1\endcsname
1075      \fi}%
1076      \def\uselanguage#1{%
1077      \input language.def
1078      \endgroup
1079      \fi
1080      \fi
1081      \chardef\l@english\z@
1082      \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.

```

1083 \def\addto#1#2{%
1084   \ifx#1\@undefined
1085     \def#1{#2}%
1086   \else
1087     \ifx#1\relax
1088       \def#1{#2}%
1089     \else
1090       {\toks@\expandafter{#1#2}%
1091        \xdef#1{\the\toks@}}%
1092     \fi
1093   \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.

```

1094 \bbl@trace{Hooks}
1095 \newcommand\AddBabelHook[3][]{%
1096   \bbl@iifunset\bbl@hk@#2{\EnableBabelHook{#2}}}%
1097   \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1098   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1099   \bbl@iifunset\bbl@ev@#2@#3@#1{%
1100     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1101     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1102   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1103 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1104 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1105 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1106 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1107   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1108   \def\bbl@elth##1{%
1109     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1110     \bbl@cs{ev@#2@}%
1111   \ifx\language\@undefined\else % Test required for Plain (?)
1112     \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1113   \def\bbl@elth##1{%
1114     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}%
1115     \bbl@cs{ev@#2@#1}%
1116   \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).

```

1117 \def\bbl@evargs{,% <- don't delete this comma

```

```

1118 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1119 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1120 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1121 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1122 beforestart=0,language=2,begindocument=1}
1123 \ifx\NewHook\@undefined\else % Test for Plain (?)
1124 \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}\NewHook{babel/*/#1}}
1125 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1126 \fi

```

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

```

1127 \providecommand\PassOptionsToLocale[2]{%
1128 \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.

```

1129 \bbl@trace{Macros for setting language files up}
1130 \def\bbl@ldfinit{%
1131 \let\bbl@screset\@empty
1132 \let\BabelStrings\bbl@opt@string
1133 \let\BabelOptions\@empty
1134 \let\BabelLanguages\relax
1135 \ifx\originalTeX\@undefined
1136 \let\originalTeX\@empty
1137 \else
1138 \originalTeX
1139 \fi}
1140 \def\LdfInit#1#2{%
1141 \chardef\atcatcode=\catcode`\@
1142 \catcode`\@=11\relax
1143 \chardef\eqcatcode=\catcode`\=
1144 \catcode`\=12\relax
1145 \ifpackagewith{babel}{ensureinfo=off}}}%
1146 {\ifx\InputIfFileExists\@undefined\else
1147 \bbl@ifunset{\bbl@lname@#1}%
1148 {{\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1149 \def\language#1}%
1150 \bbl@id@assign
1151 \bbl@load@info{#1}}}%
1152 {}%
1153 \fi}%
1154 \expandafter\if\expandafter\@backslashchar
1155 \expandafter\@car\string#2\@nil
1156 \ifx#2\@undefined\else
1157 \ldf@quit{#1}%

```

```

1158 \fi
1159 \else
1160 \expandafter\ifx\csname#2\endcsname\relax\else
1161 \ldf@quit{#1}%
1162 \fi
1163 \fi
1164 \bbl@ldfinit}

```

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

```

1165 \def\ldf@quit#1{%
1166 \expandafter\main@language\expandafter{#1}%
1167 \catcode`\@=\atcatcode \let\atcatcode\relax
1168 \catcode`\==\eqcatcode \let\eqcatcode\relax
1169 \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.

```

1170 \def\bbl@afterldf{%
1171 \bbl@afterlang
1172 \let\bbl@afterlang\relax
1173 \let\BabelModifiers\relax
1174 \let\bbl@screset\relax}%
1175 \def\ldf@finish#1{%
1176 \loadlocalcfg{#1}%
1177 \bbl@afterldf
1178 \expandafter\main@language\expandafter{#1}%
1179 \catcode`\@=\atcatcode \let\atcatcode\relax
1180 \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1181 \@onlypreamble\LdfInit
1182 \@onlypreamble\ldf@quit
1183 \@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.

```

1184 \def\main@language#1{%
1185 \def\bbl@main@language{#1}%
1186 \let\language\bbl@main@language
1187 \let\localename\bbl@main@language
1188 \let\mainlocalename\bbl@main@language
1189 \bbl@id@assign
1190 \ifcase\bbl@engine\or
1191 \ifx\setattribute\@undefined\else
1192 \setattribute\bbl@attr@locale\localeid
1193 \fi
1194 \fi
1195 \bbl@patterns{\language}

```

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

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

```

1196 \def\bbl@beforestart{%

```

```

1197 \def\@nolanerr##1{%
1198   \bbl@carg\chardef{l@##1}\z@
1199   \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1200 \bbl@usehooks{beforestart}{}%
1201 \global\let\bbl@beforestart\relax
1202 \AtBeginDocument{%
1203   {\@nameuse{bbl@beforestart}}% Group!
1204   \if@filesw
1205     \providecommand\babel@aux[2]{}%
1206     \immediate\write\@mainaux{\unexpanded{%
1207       \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1208     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1209   \fi
1210   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1211   \ifbbl@single % must go after the line above.
1212     \renewcommand\selectlanguage[1]{}%
1213     \renewcommand\foreignlanguage[2]{#2}%
1214     \global\let\babel@aux\@gobbletwo % Also as flag
1215   \fi}

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

1216 \def\select@language@x#1{%
1217   \ifcase\bbl@select@type
1218     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1219   \else
1220     \select@language{#1}%
1221   \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.

```

1222 \bbl@trace{Shorhands}
1223 \def\bbl@withactive#1#2{%
1224   \begingroup
1225     \lccode`~=#2\relax
1226     \lowercase{\endgroup#1~}}

```

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

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

```

1227 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1228   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1229   \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\makeother#1}}%
1230   \ifx\nfss@catcodes\undefined\else
1231     \begingroup
1232       \catcode`#1\active
1233       \nfss@catcodes
1234       \ifnum\catcode`#1=\active
1235         \endgroup
1236         \bbl@add\nfss@catcodes{\@makeother#1}%
1237       \else
1238         \endgroup
1239       \fi
1240   \fi}

```

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

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

```
1241 \def\bbl@active@def#1#2#3#4{%
1242   \namedef{#3#1}{%
1243     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1244       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1245     \else
1246       \bbl@afterfi\csname#2@sh@#1@\endcsname
1247     \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.

```
1248   \long\@namedef{#3@arg#1}##1{%
1249     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1250       \bbl@afterelse\csname#4#1\endcsname##1%
1251     \else
1252       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1253     \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.

```
1254 \def\@initiate@active@char#1{%
1255   \bbl@ifunset{active@char\string#1}%
1256   {\bbl@withactive
1257     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1258   {}}

```

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).

```
1259 \def\@initiate@active@char#1#2#3{%
1260   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1261   \ifx#1\@undefined
1262     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1263   \else
1264     \bbl@csarg\let{oridef@#2}#1%
1265     \bbl@csarg\edef{oridef@#2}{%
1266       \let\noexpand#1%
1267       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1268   \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`).

```
1269   \ifx#1#3\relax

```

```

1270 \expandafter\let\csname normal@char#2\endcsname#3%
1271 \else
1272 \bbl@info{Making #2 an active character}%
1273 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1274 \@namedef{normal@char#2}{%
1275 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1276 \else
1277 \@namedef{normal@char#2}{#3}%
1278 \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).

```

1279 \bbl@restoreactive{#2}%
1280 \AtBeginDocument{%
1281 \catcode`#2\active
1282 \if@files
1283 \immediate\write\@mainaux{\catcode`\string#2\active}%
1284 \fi}%
1285 \expandafter\bbl@add@special\csname#2\endcsname
1286 \catcode`#2\active
1287 \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}`).

```

1288 \let\bbl@tempa\@firstoftwo
1289 \if\string^#2%
1290 \def\bbl@tempa{\noexpand\textormath}%
1291 \else
1292 \ifx\bbl@mathnormal\@undefined\else
1293 \let\bbl@tempa\bbl@mathnormal
1294 \fi
1295 \fi
1296 \expandafter\edef\csname active@char#2\endcsname{%
1297 \bbl@tempa
1298 {\noexpand\if@safe@actives
1299 \noexpand\expandafter
1300 \expandafter\noexpand\csname normal@char#2\endcsname
1301 \noexpand\else
1302 \noexpand\expandafter
1303 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1304 \noexpand\fi}%
1305 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1306 \bbl@csarg\edef{doactive#2}{%
1307 \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

$$\text{active@prefix}\langle char\rangle\text{normal@char}\langle char\rangle$$

(where `\active@char{char}` is *one* control sequence!).

```

1308 \bbl@csarg\edef{active@#2}{%
1309 \noexpand\active@prefix\noexpand#1%
1310 \expandafter\noexpand\csname active@char#2\endcsname}%
1311 \bbl@csarg\edef{normal@#2}{%
1312 \noexpand\active@prefix\noexpand#1%
1313 \expandafter\noexpand\csname normal@char#2\endcsname}%
1314 \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.

```
1315 \bbl@active@def#2\user@group{user@active}{language@active}%
1316 \bbl@active@def#2\language@group{language@active}{system@active}%
1317 \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.

```
1318 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1319 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1320 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1321 {\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 `\prim@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.

```
1322 \if\string'#2%
1323 \let\prim@s\bbl@prim@s
1324 \let\active@math@prime#1%
1325 \fi
1326 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
1327 <(*More package options)> ≡
1328 \DeclareOption{math=active}{}
1329 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1330 <(/More package options)>
```

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package *and* the end of the *ldf*.

```
1331 \ifpackagewith{babel}{KeepShorthandsActive}%
1332 {\let\bbl@restoreactive@gobble}%
1333 {\def\bbl@restoreactive#1{%
1334 \bbl@exp{%
1335 \\\AfterBabelLanguage\\CurrentOption
1336 {\catcode`#1=\the\catcode`#1\relax}%
1337 \\\AtEndOfPackage
1338 {\catcode`#1=\the\catcode`#1\relax}}}%
1339 \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.

```
1340 \def\bbl@sh@select#1#2{%
1341 \expandafter\ifx\csname#1@sh#2@sel\endcsname\relax
1342 \bbl@afterelse\bbl@scndcs
1343 \else
1344 \bbl@afterfi\csname#1@sh#2@sel\endcsname
1345 \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 on \ifincsname is available. If there is, the expansion will be more robust.

```

1346 \begingroup
1347 \bbl@ifunset{ifincsname}
1348 {\gdef\active@prefix#1{%
1349   \ifx\protect\@typeset@protect
1350   \else
1351     \ifx\protect\@unexpandable@protect
1352       \noexpand#1%
1353     \else
1354       \protect#1%
1355     \fi
1356     \expandafter\@gobble
1357   \fi}}
1358 {\gdef\active@prefix#1{%
1359   \ifincsname
1360     \string#1%
1361     \expandafter\@gobble
1362   \else
1363     \ifx\protect\@typeset@protect
1364     \else
1365       \ifx\protect\@unexpandable@protect
1366         \noexpand#1%
1367       \else
1368         \protect#1%
1369       \fi
1370       \expandafter\expandafter\expandafter\@gobble
1371     \fi
1372   \fi}}
1373 \endgroup

```

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

```

1374 \newif\if@safe@actives
1375 \@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.

```

1376 \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.

```

1377 \chardef\bbl@activated\z@
1378 \def\bbl@activate#1{%
1379   \chardef\bbl@activated\@ne
1380   \bbl@withactive{\expandafter\let\expandafter}#1%
1381   \csname bbl@active@\string#1\endcsname}
1382 \def\bbl@deactivate#1{%
1383   \chardef\bbl@activated\tw@
1384   \bbl@withactive{\expandafter\let\expandafter}#1%

```

```
1385 \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs

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

```
1386 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1387 \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.

```
1388 \def\babel@texpdf#1#2#3#4{%
1389 \ifx\texorpdfstring\undefined
1390 \textormath{#1}{#3}%
1391 \else
1392 \texorpdfstring{\textormath{#1}{#3}}{#2}%
1393 % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1394 \fi}
1395 %
1396 \def\declare@shorthand#1#2{\@decl@short{#1}#2@nil}
1397 \def\@decl@short#1#2#3@nil#4{%
1398 \def\bbl@tempa{#3}%
1399 \ifx\bbl@tempa\empty
1400 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1401 \bbl@ifunset{#1@sh@\string#2@}{}%
1402 {\def\bbl@tempa{#4}%
1403 \expandafter\ifx\csname#1@sh@\string#2@endcsname\bbl@tempa
1404 \else
1405 \bbl@info
1406 {Redefining #1 shorthand \string#2\}%
1407 in language \CurrentOption}%
1408 \fi}%
1409 \@namedef{#1@sh@\string#2@}{#4}%
1410 \else
1411 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1412 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1413 {\def\bbl@tempa{#4}%
1414 \expandafter\ifx\csname#1@sh@\string#2@\string#3@endcsname\bbl@tempa
1415 \else
1416 \bbl@info
1417 {Redefining #1 shorthand \string#2\string#3\}%
1418 in language \CurrentOption}%
1419 \fi}%
1420 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1421 \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.

```
1422 \def\textormath{%
1423 \ifmmode
1424 \expandafter\@secondoftwo
1425 \else
1426 \expandafter\@firstoftwo
1427 \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’.

```
1428 \def\user@group{user}
1429 \def\language@group{english}
1430 \def\system@group{system}
```

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

```
1431 \def\usesshorthands{%
1432   \ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1433   \def\bbl@usesesh@s#1{%
1434     \bbl@usesesh@x
1435     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1436     {#1}}
1437   \def\bbl@usesesh@x#1#2{%
1438     \bbl@ifshorthand{#2}%
1439     {\def\user@group{user}%
1440      \initiate@active@char{#2}%
1441      #1%
1442      \bbl@activate{#2}}%
1443     {\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.

```
1444 \def\user@language@group{user@\language@group}
1445 \def\bbl@set@user@generic#1#2{%
1446   \bbl@ifunset{user@generic@active#1}%
1447   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1448    \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1449    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1450      \expandafter\noexpand\csname normal@char#1\endcsname}%
1451    \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1452      \expandafter\noexpand\csname user@active#1\endcsname}}%
1453   \@empty}
1454 \newcommand\defineshorthand[3][user]{%
1455   \edef\bbl@tempa{\zap@space#1 \@empty}%
1456   \bbl@for\bbl@tempb\bbl@tempa{%
1457     \if*\expandafter\@car\bbl@tempb\@nil
1458       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1459       \@expandtwoargs
1460       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1461     \fi
1462     \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.

```
1463 \def\languageshorthands#1{%
1464   \bbl@ifsamestring{none}{#1}{}%
1465   \bbl@once{short-\localename-#1}{%
1466     \bbl@info{'\localename' activates '#1' shorthands.\@Reported}}%
1467   \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 `\aliasshandands{"}{/}` is `\active@prefix / \active@char/`, so we still need to let the latter to `\active@char`".

```

1468 \def\aliasshorthand#1#2{%
1469   \bbl@ifshorthand{#2}%
1470   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1471     \ifx\document\@notprerr
1472       \@notshorthand{#2}%
1473     \else
1474       \initiate@active@char{#2}%
1475       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1476       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1477       \bbl@activate{#2}%
1478     \fi
1479   \fi}%
1480   {\bbl@error{shorthand-is-off}{#2}{}}}
```

\@notshorthand

```

1481 \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.

```

1482 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1483 \DeclareRobustCommand*\shorthandoff{%
1484   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1485 \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.

```

1486 \def\bbl@switch@sh#1#2{%
1487   \ifx#2\@nnil\else
1488     \bbl@ifunset{bbl@active@\string#2}%
1489     {\bbl@error{not-a-shorthand-b}{#2}{}}%
1490     {\ifcase#1%   off, on, off*
1491       \catcode`#2\relax
1492     \or
1493       \catcode`#2\active
1494       \bbl@ifunset{bbl@shdef@\string#2}%
1495       {}%
1496       {\bbl@withactive{\expandafter\let\expandafter}#2%
1497         \csname bbl@shdef@\string#2\endcsname
1498         \bbl@csarg\let{shdef@\string#2}\relax}%
1499       \ifcase\bbl@activated\or
1500         \bbl@activate{#2}%
1501       \else
1502         \bbl@deactivate{#2}%
1503       \fi
1504     \or
1505       \bbl@ifunset{bbl@shdef@\string#2}%
1506       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1507       {}%
1508       \csname bbl@oricat@\string#2\endcsname
1509       \csname bbl@oridef@\string#2\endcsname
1510     \fi}%

```

```

1511 \bbl@afterfi\bbl@switch@sh#1%
1512 \fi}

```

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

```

1513 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1514 \def\bbl@putsh#1{%
1515 \bbl@ifunset{\bbl@active@\string#1}%
1516 {\bbl@putsh@i#1\@empty\@nnil}%
1517 {\csname bbl@active@\string#1\endcsname}}
1518 \def\bbl@putsh@i#1#2\@nnil{%
1519 \csname\language@group @sh@\string#1@%
1520 \ifx\@empty#2\else\string#2@\fi\endcsname}
1521 %
1522 \ifx\bbl@opt@shorthands\@nnil\else
1523 \let\bbl@s@initiate@active@char@initiate@active@char
1524 \def\initiate@active@char#1{%
1525 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1526 \let\bbl@s@switch@sh\bbl@switch@sh
1527 \def\bbl@switch@sh#1#2{%
1528 \ifx#2\@nnil\else
1529 \bbl@afterfi
1530 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1531 \fi}
1532 \let\bbl@s@activate\bbl@activate
1533 \def\bbl@activate#1{%
1534 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1535 \let\bbl@s@deactivate\bbl@deactivate
1536 \def\bbl@deactivate#1{%
1537 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1538 \fi

```

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

```

1539 \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.

```

1540 \def\bbl@prim@s{%
1541 \prime\futurelet\@let@token\bbl@pr@m@s}
1542 \def\bbl@if@primes#1#2{%
1543 \ifx#1\@let@token
1544 \expandafter\@firstoftwo
1545 \else\ifx#2\@let@token
1546 \bbl@afterelse\expandafter\@firstoftwo
1547 \else
1548 \bbl@afterfi\expandafter\@secondoftwo
1549 \fi\fi}
1550 \begingroup
1551 \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1552 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"='\^
1553 \lowercase{%
1554 \gdef\bbl@pr@m@s{%
1555 \bbl@if@primes" '%
1556 \pr@@@s
1557 {\bbl@if@primes*\^{\pr@@@t\egroup}}}
1558 \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).

```
1559 \initiate@active@char{~}
1560 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1561 \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.

```
1562 \expandafter\def\csname OT1dqpos\endcsname{127}
1563 \expandafter\def\csname T1dqpos\endcsname{4}
```

When the macro \f@encoding is undefined (as it is in plain T_EX) we define it here to expand to OT1

```
1564 \ifx\f@encoding\undefined
1565   \def\f@encoding{OT1}
1566 \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.

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

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

```
1567 \bbl@trace{Language attributes}
1568 \newcommand\languageattribute[2]{%
1569   \def\bbl@tempc{#1}%
1570   \bbl@fixname\bbl@tempc
1571   \bbl@iflanguage\bbl@tempc{%
1572     \bbl@vforeach{#2}{%
1573       \ifx\bbl@known@attrs\undefined
1574         \in@false
1575       \else
1576         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1577       \fi
1578       \ifin@
1579         \bbl@warning{%
1580           You have more than once selected the attribute '##1'\%
1581           for language #1. Reported}%
1582       \else
1583         \bbl@info{Activated '##1' attribute for\\%
1584           '\bbl@tempc'. Reported}%
1585         \bbl@exp{%
1586           \\ \bbl@add@list\\ \bbl@known@attrs{\bbl@tempc-##1}}%
1587         \edef\bbl@tempa{\bbl@tempc-##1}%
1588         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes
1589         {\csname\bbl@tempc @attr##1\endcsname}%
1590         {\bbl@error{unknown-attribute}{\bbl@tempc}{##1}}}%
1591       \fi}}
1592 \@onlypreamble\languageattribute
```

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

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

```
1593 \def\bbl@declare@ttribute#1#2#3{%
1594   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1595   \ifin@
1596     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1597   \fi
1598   \bbl@add@list\bbl@attributes{#1-#2}%
1599   \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.

```
1600 \def\bbl@ifattributeset#1#2#3#4{%
1601   \ifx\bbl@known@attribs\@undefined
1602     \in@false
1603   \else
1604     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1605   \fi
1606   \ifin@
1607     \bbl@afterelse#3%
1608   \else
1609     \bbl@afterfi#4%
1610   \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.

```
1611 \def\bbl@ifknown@ttrib#1#2{%
1612   \let\bbl@tempa\@secondoftwo
1613   \bbl@loopx\bbl@tempb{#2}{%
1614     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1615   \ifin@
1616     \let\bbl@tempa\@firstoftwo
1617   \else
1618     \fi}%
1619   \bbl@tempa}
```

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

```
1620 \def\bbl@clear@ttribs{%
1621   \ifx\bbl@attributes\@undefined\else
1622     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1623       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1624     \let\bbl@attributes\@undefined
1625   \fi}
1626 \def\bbl@clear@ttrib#1-#2.{%
1627   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1628 \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.

```
1629 \bbl@trace{Macros for saving definitions}
1630 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1631 \newcount\babel@savecnt
1632 \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.

```
1633 \def\babel@save#1{%
1634   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1635   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1636     \expandafter{\expandafter,\bbl@savextras,}}%
1637   \expandafter\in@\bbl@tempa
1638   \ifin@%else
1639     \bbl@add\bbl@savextras{,{#1,}}%
1640     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1641     \toks@{\expandafter{\originalTeX\let#1=}}%
1642     \bbl@exp{%
1643       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1644     \advance\babel@savecnt@ne
1645   \fi}
1646 \def\babel@savevariable#1{%
1647   \toks@{\expandafter{\originalTeX #1=}}%
1648   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```
1649 \def\bbl@redefine#1{%
1650   \edef\bbl@tempa{\bbl@stripslash#1}%
1651   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1652   \expandafter\def\csname\bbl@tempa\endcsname}
1653 \@onlypreamble\bbl@redefine
```

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

```
1654 \def\bbl@redefine@long#1{%
1655   \edef\bbl@tempa{\bbl@stripslash#1}%
1656   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1657   \long\expandafter\def\csname\bbl@tempa\endcsname}
1658 \@onlypreamble\bbl@redefine@long
```

\bbl@redefineroobust For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo`. So it is necessary to check whether `\foo` exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define `\foo`.

```

1659 \def\bbl@redefineroobust#1{%
1660   \edef\bbl@tempa{\bbl@stripslash#1}%
1661   \bbl@iifunset{\bbl@tempa\space}%
1662   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1663    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1664   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1665   \@namedef{\bbl@tempa\space}}
1666 \@onlypreamble\bbl@redefineroobust

```

4.11. French spacing

\bbl@frenchspacing

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

```

1667 \def\bbl@frenchspacing{%
1668   \ifnum\the\sfcode`\.\=\@m
1669     \let\bbl@nonfrenchspacing\relax
1670   \else
1671     \frenchspacing
1672     \let\bbl@nonfrenchspacing\nonfrenchspacing
1673   \fi}
1674 \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.

```

1675 \let\bbl@elt\relax
1676 \edef\bbl@fs@chars{%
1677   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1678   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1679   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1680 \def\bbl@pre@fs{%
1681   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1682   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1683 \def\bbl@post@fs{%
1684   \bbl@save@sfcodes
1685   \edef\bbl@tempa{\bbl@cl{frspc}}%
1686   \edef\bbl@tempa{\expandafter\car\bbl@tempa\@nil}%
1687   \if u\bbl@tempa      % do nothing
1688   \else\if n\bbl@tempa % non french
1689     \def\bbl@elt##1##2##3{%
1690       \ifnum\sfcode`##1=##2\relax
1691       \babel@savevariable{\sfcode`##1}%
1692       \sfcode`##1=##3\relax
1693     \fi}%
1694     \bbl@fs@chars
1695   \else\if y\bbl@tempa % french
1696     \def\bbl@elt##1##2##3{%
1697       \ifnum\sfcode`##1=##3\relax
1698       \babel@savevariable{\sfcode`##1}%
1699       \sfcode`##1=##2\relax
1700     \fi}%
1701     \bbl@fs@chars
1702   \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.

```

1703 \bbl@trace{Hyphens}
1704 \@onlypreamble\babelhyphenation
1705 \AtEndOfPackage{%
1706   \newcommand\babelhyphenation[2][\@empty]{%
1707     \ifx\bbl@hyphenation@\relax
1708       \let\bbl@hyphenation@\@empty
1709     \fi
1710     \ifx\bbl@hyphlist@\@empty\else
1711       \bbl@warning{%
1712         You must not intermingle \string\selectlanguage\space and\\%
1713         \string\babelhyphenation\space or some exceptions will not\\%
1714         be taken into account. Reported}%
1715       \fi
1716       \ifx\@empty#1%
1717         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1718       \else
1719         \bbl@vforeach{#1}{%
1720           \def\bbl@tempa{##1}%
1721           \bbl@fixname\bbl@tempa
1722           \bbl@iflanguage\bbl@tempa{%
1723             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1724               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1725               {}%
1726               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1727               #2}}}%
1728         \fi}}

```

\babelhyphenmins Only L^AT_EX (basically because it's defined with a L^AT_EX tool).

```

1729 \ifx\NewDocumentCommand\@undefined\else
1730   \NewDocumentCommand\babelhyphenmins{sommo}{%
1731     \IfNoValueTF{#2}%
1732       {\protected@edef\bbl@hyphenmins@{\set@hyphenmins{#3}{#4}}}%
1733       \IfValueT{#5}{%
1734         \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1735       \IfBooleanT{#1}{%
1736         \lefthyphenmin=#3\relax
1737         \righthyphenmin=#4\relax
1738         \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1739     {\edef\bbl@tempb{\zap@space#2 \@empty}%
1740      \bbl@for\bbl@tempa\bbl@tempb{%
1741        \@namedef{bbl@hyphenmins@\bbl@tempa}{\set@hyphenmins{#3}{#4}}}%
1742      \IfValueT{#5}{%
1743        \@namedef{bbl@hyphenatmin@\bbl@tempa}{\hyphenationmin=#5\relax}}}%
1744      \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}}}%
1745   \fi

```

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

```

1746 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1747 \def\bbl@t@one{Tl}
1748 \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.

```

1749 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1750 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1751 \def\bbl@hyphen{%

```

```

1752 \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1753 \def\bbl@hyphen@i#1#2{%
1754 \lowercase{\bbl@ifunset{\bbl@hy#1#2\@empty}}}%
1755 {\csname bbl@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1756 {\lowercase{\csname bbl@hy#1#2\@empty\endcsname}}}

```

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

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

```

1757 \def\bbl@usehyphen#1{%
1758 \leavevmode
1759 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1760 \nobreak\hskip\z@skip}
1761 \def\bbl@usehyphen#1{%
1762 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1763 \def\bbl@hyphenchar{%
1764 \ifnum\hyphenchar\font=\m@ne
1765 \babe\nullhyphen
1766 \else
1767 \char\hyphenchar\font
1768 \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.

```

1769 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1770 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1771 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1772 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1773 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1774 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1775 \def\bbl@hy@repeat{%
1776 \bbl@usehyphen{
1777 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1778 \def\bbl@hy@repeat{%
1779 \bbl@usehyphen{
1780 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1781 \def\bbl@hy@empty{\hskip\z@skip}
1782 \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.

```

1783 \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.

```

1784 \bbl@trace{Multiencoding strings}
1785 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1786 ({*More package options}) ≡
1787 \DeclareOption{nocase}{}
1788 ({/More package options})

```

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

```

1789 ({*More package options}) ≡
1790 \let\bbl@opt@strings\@nnil % accept strings=value
1791 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1792 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1793 \def\BabelStringsDefault{generic}
1794 ({/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.

```

1795 \@onlypreamble\StartBabelCommands
1796 \def\StartBabelCommands{%
1797   \begingroup
1798   \@tempcnta="7F
1799   \def\bbl@tempa{%
1800     \ifnum\@tempcnta>"FF\else
1801       \catcode\@tempcnta=11
1802       \advance\@tempcnta\@ne
1803       \expandafter\bbl@tempa
1804     \fi}%
1805   \bbl@tempa
1806   <@Macros local to BabelCommands@>
1807   \def\bbl@provstring##1##2{%
1808     \providecommand##1{##2}%
1809     \bbl@tglobal##1}%
1810   \global\let\bbl@scafter\@empty
1811   \let\StartBabelCommands\bbl@startcmds
1812   \ifx\BabelLanguages\relax
1813     \let\BabelLanguages\CurrentOption
1814   \fi
1815   \begingroup
1816   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1817   \StartBabelCommands}
1818 \def\bbl@startcmds{%
1819   \ifx\bbl@screset\@nnil\else
1820     \bbl@usehooks{stopcommands}{}%
1821   \fi
1822   \endgroup
1823   \begingroup
1824   \@ifstar
1825   {\ifx\bbl@opt@strings\@nnil
1826     \let\bbl@opt@strings\BabelStringsDefault
1827   \fi
1828   \bbl@startcmds@i}%
1829   \bbl@startcmds@i}
1830 \def\bbl@startcmds@i#1#2{%
1831   \edef\bbl@L{\zap@space#1 \@empty}%
1832   \edef\bbl@G{\zap@space#2 \@empty}%
1833   \bbl@startcmds@ii}
1834 \let\bbl@startcmds\StartBabelCommands

```

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

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

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

```

1835 \newcommand\bbl@startcmds@ii[1][\@empty]{%

```

```

1836 \let\SetString\@gobbletwo
1837 \let\bbl@stringdef\@gobbletwo
1838 \let\AfterBabelCommands\@gobble
1839 \ifx\@empty#1%
1840   \def\bbl@sc@label{generic}%
1841   \def\bbl@encstring##1##2{%
1842     \ProvideTextCommandDefault##1{##2}%
1843     \bbl@tglobal##1%
1844     \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1845   \let\bbl@sctest\in@true
1846 \else
1847   \let\bbl@sc@charset\space % <- zapped below
1848   \let\bbl@sc@fontenc\space % <- " "
1849   \def\bbl@tempa##1=##2\@nil{%
1850     \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1851   \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1852   \def\bbl@tempa##1 ##2{% space -> comma
1853     ##1%
1854     \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1855   \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1856   \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1857   \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1858   \def\bbl@encstring##1##2{%
1859     \bbl@foreach\bbl@sc@fontenc{%
1860       \bbl@ifunset{T@###1}%
1861       {}%
1862       {\ProvideTextCommand##1{###1}{##2}%
1863         \bbl@tglobal##1%
1864         \expandafter
1865         \bbl@tglobal\csname###1\string##1\endcsname}}}%
1866   \def\bbl@sctest{%
1867     \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
1868 \fi
1869 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1870 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1871   \let\AfterBabelCommands\bbl@aftercmds
1872   \let\SetString\bbl@setstring
1873   \let\bbl@stringdef\bbl@encstring
1874 \else % i.e., strings=value
1875   \bbl@sctest
1876 \fin@
1877   \let\AfterBabelCommands\bbl@aftercmds
1878   \let\SetString\bbl@setstring
1879   \let\bbl@stringdef\bbl@provstring
1880 \fi\fi\fi
1881 \bbl@scswitch
1882 \ifx\bbl@G\@empty
1883   \def\SetString##1##2{%
1884     \bbl@error{missing-group}{##1}{}}%
1885 \fi
1886 \ifx\@empty#1%
1887   \bbl@usehooks{defaultcommands}{}%
1888 \else
1889   \@expandtwoargs
1890   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1891 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing.

The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has

been loaded).

```

1892 \def\bbl@forlang#1#2{%
1893   \bbl@for#1\bbl@L{%
1894     \bbl@xin@{,#1,},{,\BabelLanguages,}%
1895     \ifin@#2\relax\fi}}
1896 \def\bbl@scswitch{%
1897   \bbl@forlang\bbl@tempa{%
1898     \ifx\bbl@G\@empty\else
1899       \ifx\SetString@gobbletwo\else
1900         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1901         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1902         \ifin@\else
1903           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1904           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1905         \fi
1906       \fi
1907     \fi}}
1908 \AtEndOfPackage{%
1909   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1910   \let\bbl@scswitch\relax}
1911 \@onlypreamble\EndBabelCommands
1912 \def\EndBabelCommands{%
1913   \bbl@usehooks{stopcommands}{}%
1914   \endgroup
1915   \endgroup
1916   \bbl@scafter}
1917 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

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

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

```

1918 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1919   \bbl@forlang\bbl@tempa{%
1920     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1921     \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1922     {\bbl@exp{%
1923       \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1924     }%
1925     \def\BabelString{#2}%
1926     \bbl@usehooks{stringprocess}{}%
1927     \expandafter\bbl@stringdef
1928     \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.

```

1929 \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.

```

1930 ((*Macros local to BabelCommands)) ≡
1931 \def\SetStringLoop##1#2{%
1932   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1933   \count@\z@
1934   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1935     \advance\count@\@ne
1936     \toks@\expandafter{\bbl@tempa}%
1937     \bbl@exp{%
1938       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%

```

```

1939 \count@=\the\count@\relax}}}%
1940 <</Macros local to BabelCommands>>

```

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

```

1941 \def\bbl@aftercmds#1{%
1942 \toks@\expandafter{\bbl@scafter#1}%
1943 \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.

```

1944 <<*Macros local to BabelCommands>> ≡
1945 \newcommand\SetCase[3][]{%
1946 \def\bbl@tempa####1####2{%
1947 \ifx####1\@empty\else
1948 \bbl@carg\bbl@add{extras\CurrentOption}{%
1949 \bbl@carg\babel@save{c__text_uppercase_\string####1_tl}%
1950 \bbl@carg\def{c__text_uppercase_\string####1_tl}{####2}%
1951 \bbl@carg\babel@save{c__text_lowercase_\string####2_tl}%
1952 \bbl@carg\def{c__text_lowercase_\string####2_tl}{####1}}%
1953 \expandafter\bbl@tempa
1954 \fi}%
1955 \bbl@tempa##1\@empty\@empty
1956 \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1957 <</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.

```

1958 <<*Macros local to BabelCommands>> ≡
1959 \newcommand\SetHyphenMap[1]{%
1960 \bbl@forlang\bbl@tempa{%
1961 \expandafter\bbl@stringdef
1962 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1963 <</Macros local to BabelCommands>>

```

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

```

1964 \newcommand\BabelLower[2]{% one to one.
1965 \ifnum\lccode#1=#2\else
1966 \babel@savevariable{\lccode#1}%
1967 \lccode#1=#2\relax
1968 \fi}
1969 \newcommand\BabelLowerMM[4]{% many-to-many
1970 \@tempcnta=#1\relax
1971 \@tempcntb=#4\relax
1972 \def\bbl@tempa{%
1973 \ifnum\@tempcnta>#2\else
1974 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1975 \advance\@tempcnta#3\relax
1976 \advance\@tempcntb#3\relax
1977 \expandafter\bbl@tempa
1978 \fi}%
1979 \bbl@tempa}
1980 \newcommand\BabelLowerM0[4]{% many-to-one
1981 \@tempcnta=#1\relax
1982 \def\bbl@tempa{%
1983 \ifnum\@tempcnta>#2\else
1984 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1985 \advance\@tempcnta#3
1986 \expandafter\bbl@tempa
1987 \fi}%
1988 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1989 ({*More package options}) ≡
1990 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1991 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1992 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1993 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1994 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1995 (/More package options)

```

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

```

1996 \AtEndOfPackage{%
1997   \ifx\bbl@opt@hyphenmap\undefined
1998     \bbl@xin@{,}{\bbl@language@opts}%
1999     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2000   \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.

```

2001 \newcommand\setlocalecaption{%
2002   \ifstar\bbl@setcaption@s\bbl@setcaption@x}
2003 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2004   \bbl@trim@def\bbl@tempa{#2}%
2005   \bbl@xin@{.template}{\bbl@tempa}%
2006   \ifin@
2007     \bbl@ini@captions@template{#3}{#1}%
2008   \else
2009     \edef\bbl@tempd{%
2010       \expandafter\expandafter\expandafter
2011       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2012     \bbl@xin@
2013       {\expandafter\string\csname #2name\endcsname}%
2014       {\bbl@tempd}%
2015     \ifin@ % Renew caption
2016       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2017     \ifin@
2018       \bbl@exp{%
2019         \\bbl@ifsamestring{\bbl@tempa}{\language name}%
2020         {\bbl@scset\<#2name>\<#1#2name>}%
2021         {}}%
2022       \else % Old way converts to new way
2023         \bbl@ifunset{#1#2name}%
2024         {\bbl@exp{%
2025           \\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2026           \\bbl@ifsamestring{\bbl@tempa}{\language name}%
2027           {\def\<#2name>\<#1#2name>}}%
2028           {}}}%
2029       {}%
2030     \fi
2031   \else
2032     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2033     \ifin@ % New way
2034     \bbl@exp{%
2035       \\bbl@add\<captions#1>\bbl@scset\<#2name>\<#1#2name>}%
2036       \\bbl@ifsamestring{\bbl@tempa}{\language name}%
2037       {\bbl@scset\<#2name>\<#1#2name>}%
2038       {}}%
2039     \else % Old way, but defined in the new way
2040     \bbl@exp{%
2041       \\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2042       \\bbl@ifsamestring{\bbl@tempa}{\language name}%

```

```

2043      {\def\<#2name>{\<#1#2name>}}%
2044      {}}%
2045      \fi%
2046      \fi
2047      \@namedef{#1#2name}{#3}%
2048      \toks@{\expandafter{\bbl@captionslist}%
2049      \bbl@exp{\in{\<#2name>}{\the\toks@}}}%
2050      \ifin@else
2051      \bbl@exp{\bbl@add{\bbl@captionslist{\<#2name>}}}%
2052      \bbl@tglobal\bbl@captionslist
2053      \fi
2054      \fi}

```

4.15. Making glyphs available

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

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

```

2055 \bbl@trace{Macros related to glyphs}
2056 \def\set@low@box#1{\setbox\tw@{\hbox{,}}\setbox\z@{\hbox{#1}}%
2057   \dimen\z@{\ht\z@ \advance\dimen\z@ -\ht\tw@}%
2058   \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.

```

2059 \def\save@sf@q#1{\leavevmode
2060   \begingroup
2061   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2062   \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.

```

2063 \ProvideTextCommand{\quotedblbase}{OT1}{%
2064   \save@sf@q{\set@low@box{\textquotedblright/}}%
2065   \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.

```

2066 \ProvideTextCommandDefault{\quotedblbase}{%
2067   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2068 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2069   \save@sf@q{\set@low@box{\textquoteright/}}%
2070   \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.

```

2071 \ProvideTextCommandDefault{\quotesinglbase}{%
2072   \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```
2073 \ProvideTextCommand{\guillemetleft}{OT1}{%
2074   \ifmmode
2075     \ll
2076   \else
2077     \save@sf@q{\nobreak
2078       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2079   \fi}
2080 \ProvideTextCommand{\guillemetright}{OT1}{%
2081   \ifmmode
2082     \gg
2083   \else
2084     \save@sf@q{\nobreak
2085       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2086   \fi}
2087 \ProvideTextCommand{\guillemotleft}{OT1}{%
2088   \ifmmode
2089     \ll
2090   \else
2091     \save@sf@q{\nobreak
2092       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2093   \fi}
2094 \ProvideTextCommand{\guillemotright}{OT1}{%
2095   \ifmmode
2096     \gg
2097   \else
2098     \save@sf@q{\nobreak
2099       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2100   \fi}
```

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

```
2101 \ProvideTextCommandDefault{\guillemetleft}{%
2102   \UseTextSymbol{OT1}{\guillemetleft}}
2103 \ProvideTextCommandDefault{\guillemetright}{%
2104   \UseTextSymbol{OT1}{\guillemetright}}
2105 \ProvideTextCommandDefault{\guillemotleft}{%
2106   \UseTextSymbol{OT1}{\guillemotleft}}
2107 \ProvideTextCommandDefault{\guillemotright}{%
2108   \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft

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

```
2109 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2110   \ifmmode
2111     <%
2112   \else
2113     \save@sf@q{\nobreak
2114       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2115   \fi}
2116 \ProvideTextCommand{\guilsinglright}{OT1}{%
2117   \ifmmode
2118     >%
2119   \else
2120     \save@sf@q{\nobreak
2121       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2122   \fi}
```

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

```
2123 \ProvideTextCommandDefault{\guilsinglleft}{%
2124   \UseTextSymbol{OT1}{\guilsinglleft}}
2125 \ProvideTextCommandDefault{\guilsinglright}{%
2126   \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.

```
2127 \DeclareTextCommand{\ij}{OT1}{%
2128   i\kern-0.02em\bbl@allowhyphens j}
2129 \DeclareTextCommand{\IJ}{OT1}{%
2130   I\kern-0.02em\bbl@allowhyphens J}
2131 \DeclareTextCommand{\ij}{T1}{\char188}
2132 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2133 \ProvideTextCommandDefault{\ij}{%
2134   \UseTextSymbol{OT1}{\ij}}
2135 \ProvideTextCommandDefault{\IJ}{%
2136   \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).

```
2137 \def\crrtic@{\hrule height0.1ex width0.3em}
2138 \def\crrtic@{\hrule height0.1ex width0.33em}
2139 \def\ddj@{%
2140   \setbox0\hbox{d}\dimen@=\ht0
2141   \advance\dimen@lex
2142   \dimen@.45\dimen@
2143   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2144   \advance\dimen@ii.5ex
2145   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2146 \def\DDJ@{%
2147   \setbox0\hbox{D}\dimen@=.55\ht0
2148   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2149   \advance\dimen@ii.15ex % correction for the dash position
2150   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2151   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2152   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2153 %
2154 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2155 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2156 \ProvideTextCommandDefault{\dj}{%
2157   \UseTextSymbol{OT1}{\dj}}
2158 \ProvideTextCommandDefault{\DJ}{%
2159   \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.

```
2160 \DeclareTextCommand{\SS}{OT1}{SS}
2161 \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.

```
2162 \ProvideTextCommandDefault{\glq}{%  
2163 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2164 \ProvideTextCommand{\grq}{T1}{%  
2165 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}  
2166 \ProvideTextCommand{\grq}{TU}{%  
2167 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}  
2168 \ProvideTextCommand{\grq}{OT1}{%  
2169 \save@sf@q{\kern-.0125em  
2170 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%  
2171 \kern.07em\relax}}  
2172 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq

\grqq The ‘german’ double quotes.

```
2173 \ProvideTextCommandDefault{\glqq}{%  
2174 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2175 \ProvideTextCommand{\grqq}{T1}{%  
2176 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}  
2177 \ProvideTextCommand{\grqq}{TU}{%  
2178 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}  
2179 \ProvideTextCommand{\grqq}{OT1}{%  
2180 \save@sf@q{\kern-.07em  
2181 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%  
2182 \kern.07em\relax}}  
2183 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq

\frq The ‘french’ single guillemets.

```
2184 \ProvideTextCommandDefault{\flq}{%  
2185 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}  
2186 \ProvideTextCommandDefault{\frq}{%  
2187 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq

\frqq The ‘french’ double guillemets.

```
2188 \ProvideTextCommandDefault{\flqq}{%  
2189 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}  
2190 \ProvideTextCommandDefault{\frqq}{%  
2191 \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).

```

2192 \def\umlauthigh{%
2193   \def\bbbl@umlauta##1{\leavevmode\bgroup%
2194     \accent\csname\fontencoding dqpos\endcsname
2195     ##1\bbbl@allowhyphens\egroup}%
2196   \let\bbbl@umlaute\bbbl@umlauta}
2197 \def\umlautlow{%
2198   \def\bbbl@umlauta{\protect\lower@umlaut}}
2199 \def\umlautelowlow{%
2200   \def\bbbl@umlaute{\protect\lower@umlaut}}
2201 \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.

```

2202 \expandafter\ifx\csname U@D\endcsname\relax
2203   \csname newdimen\endcsname U@D
2204 \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.

```

2205 \def\lower@umlaut#1{%
2206   \leavevmode\bgroup
2207     \U@D lex%
2208     {\setbox\z@\hbox{%
2209       \char\csname\fontencoding dqpos\endcsname}%
2210       \dimen@ -.45ex\advance\dimen@ht\z@
2211       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2212     \accent\csname\fontencoding dqpos\endcsname
2213     \fontdimen5\font\U@D #1%
2214   \egroup}

```

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

```

2215 \AtBeginDocument{%
2216   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbbl@umlauta{a}}%
2217   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbbl@umlaute{e}}%
2218   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbbl@umlaute{i}}%
2219   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbbl@umlaute{i}}%
2220   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbbl@umlauta{o}}%
2221   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbbl@umlauta{u}}%
2222   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbbl@umlauta{A}}%
2223   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbbl@umlaute{E}}%
2224   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbbl@umlaute{I}}%
2225   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbbl@umlauta{O}}%
2226   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbbl@umlauta{U}}%

```

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

```

2227 \ifx\l@english\undefined
2228   \chardef\l@english\z@
2229 \fi

```

```

2230 % The following is used to cancel rules in ini files (see Amharic).
2231 \ifx\l@unhyphenated\@undefined
2232   \newlanguage\l@unhyphenated
2233 \fi

```

4.16. Layout

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

```

2234 \bbl@trace{Bidi layout}
2235 \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.

```

2236 \bbl@trace{Input engine specific macros}
2237 \ifcase\bbl@engine
2238   \input txtbabel.def
2239 \or
2240   \input luababel.def
2241 \or
2242   \input xebabel.def
2243 \fi
2244 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2245 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2246 \ifx\babelposthyphenation\@undefined
2247   \let\babelposthyphenation\babelprehyphenation
2248   \let\babelpatterns\babelprehyphenation
2249   \let\babelcharproperty\babelprehyphenation
2250 \fi
2251 (/package|core)

```

4.18. Creating and modifying languages

Continue with $\TeX$ 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.

```

2252 (*package)
2253 \bbl@trace{Creating languages and reading ini files}
2254 \let\bbl@extend@ini\@gobble
2255 \newcommand\babelprovide[2][]{%
2256   \let\bbl@savelangname\languagename
2257   \edef\bbl@savelocaleid{\the\localeid}%
2258   \global\let\bbl@afterload\@empty
2259   % Set name and locale id
2260   \edef\languagename{#2}%
2261   \bbl@id@assign
2262   % Initialize keys
2263   \bbl@vforeach{captions,date,import,main,script,language,%
2264     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2265     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2266     Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2267     @import}%
2268     {\bbl@csarg\let{KVP@##1}\@nnil}%
2269   \global\let\bbl@release@transforms\@empty
2270   \global\let\bbl@release@casing\@empty
2271   \let\bbl@calendars\@empty
2272   \global\let\bbl@inidata\@empty
2273   \global\let\bbl@extend@ini\@gobble
2274   \global\let\bbl@included@inis\@empty
2275   \gdef\bbl@key@list{;}%

```

```

2276 \bbl@ifunset{bbl@passto@#2}%
2277   {\def\bbl@tempa{#1}}%
2278   {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}%
2279 \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2280   \in@{/}{#1}% With /, (re)sets a value in the ini
2281   \ifin@
2282     \bbl@renewinikey##1\@{##2}%
2283   \else
2284     \bbl@csarg\ifx{KVP@##1}\@nnil\else
2285       \bbl@error{unknown-provide-key}{#1}{}}%
2286     \fi
2287     \bbl@csarg\def{KVP@##1}{##2}%
2288   \fi}%
2289 \chardef\bbl@howloaded= 0:none; 1:ldf without ini; 2:ini
2290 \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2291 % == init ==
2292 \ifx\bbl@screset\undefined
2293   \bbl@ldfinit
2294 \fi
2295 % ==
2296 % If there is no import (last wins), use @import (internal, there
2297 % must be just one). To consider any order (because
2298 % \PassOptionsToLocale).
2299 \ifx\bbl@KVP@import\@nnil
2300   \let\bbl@KVP@import\bbl@KVP@@import
2301 \fi
2302 % == date (as option) ==
2303 % \ifx\bbl@KVP@date\@nnil\else
2304 % \fi
2305 % ==
2306 \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2307 \ifcase\bbl@howloaded
2308   \let\bbl@lbkflag\@empty % new
2309 \else
2310   \ifx\bbl@KVP@hyphenrules\@nnil\else
2311     \let\bbl@lbkflag\@empty
2312   \fi
2313   \ifx\bbl@KVP@import\@nnil\else
2314     \let\bbl@lbkflag\@empty
2315   \fi
2316 \fi
2317 % == import, captions ==
2318 \ifx\bbl@KVP@import\@nnil\else
2319   \bbl@exp{\\bbl@ifblank{\bbl@KVP@import}}%
2320   {\ifx\bbl@initoload\relax
2321     \begingroup
2322       \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2323       \bbl@input@texini{##2}%
2324     \endgroup
2325   \else
2326     \xdef\bbl@KVP@import{\bbl@initoload}%
2327   \fi}%
2328   {}%
2329   \let\bbl@KVP@date\@empty
2330 \fi
2331 \let\bbl@KVP@captions@@\bbl@KVP@captions
2332 \ifx\bbl@KVP@captions\@nnil
2333   \let\bbl@KVP@captions\bbl@KVP@import
2334 \fi
2335 % ==
2336 \ifx\bbl@KVP@transforms\@nnil\else
2337   \bbl@replace\bbl@KVP@transforms{ }{,}%
2338 \fi

```

```

2339 % Load ini
2340 % -----
2341 \ifcase\bbl@howloaded
2342   \bbl@provide@new{#2}%
2343 \else
2344   \bbl@ifblank{#1}%
2345   {}% With \bbl@load@basic below
2346   {\bbl@provide@renew{#2}}%
2347 \fi
2348 % Post tasks
2349 % -----
2350 % == subsequent calls after the first provide for a locale ==
2351 \ifx\bbl@inidata\@empty\else
2352   \bbl@extend@ini{#2}%
2353 \fi
2354 % ==
2355 \ifx\bbl@KVP@mapdot\@nnil\else
2356   \def\bbl@tempa{\@empty}%
2357   \ifx\bbl@KVP@mapdot\bbl@tempa\else
2358     \bbl@exp{\gdef\<bbl@map@. .@ \language\>{\[bbl@KVP@mapdot]}}%
2359   \fi
2360 \fi
2361 % == ensure captions ==
2362 \ifx\bbl@KVP@captions\@nnil\else
2363   \bbl@ifunset{bbl@extracaps@#2}%
2364   {\bbl@exp{\[bbl@babelensure[exclude=\\today]{#2}}}%
2365   {\bbl@exp{\[bbl@babelensure[exclude=\\today,
2366     include=\[bbl@extracaps@#2]]{#2}}}%
2367   \let\bbl@tempc\language
2368   \bbl@replace\bbl@tempc{-}{@}%
2369   \bbl@ifunset{bbl@ensure@\bbl@tempc}%
2370   {\bbl@exp{%
2371     \\DeclareRobustCommand\<bbl@ensure@\bbl@tempc>[1]{%
2372       \\foreignlanguage{\language}%
2373       {###1}}}%
2374   }%
2375   \bbl@exp{%
2376     \\bbl@tglobal\<bbl@ensure@\bbl@tempc>%
2377     \\bbl@tglobal\<bbl@ensure@\bbl@tempc\space>%
2378   \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.

```

2379 \bbl@load@basic{#2}%
2380 % == script, language ==
2381 % Override the values from ini or defines them
2382 \ifx\bbl@KVP@script\@nnil\else
2383   \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2384 \fi
2385 \ifx\bbl@KVP@language\@nnil\else
2386   \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2387 \fi
2388 \ifcase\bbl@engine\or
2389   \bbl@ifunset{bbl@chrng@\language}%
2390   {\directlua{
2391     Babel.set_chranges_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2392 \fi
2393 % == Line breaking: intraspace, intrapenalty ==
2394 % For CJK, East Asian, Southeast Asian, if interspace in ini
2395 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2396   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2397 \fi

```

```

2398 \bbl@provide@intraspace
2399 % == Line breaking: justification ==
2400 \ifx\bbl@KVP@justification\@nnil\else
2401   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2402 \fi
2403 \ifx\bbl@KVP@linebreaking\@nnil\else
2404   \bbl@xin@{\,\bbl@KVP@linebreaking,}%
2405   {,elongated,kashida,cjk,padding,unhyphenated,}%
2406 \ifin@
2407   \bbl@csarg\xdef
2408   {\lnbrk@{\language\name}}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2409 \fi
2410 \fi
2411 \bbl@xin@{/e}{\bbl@cl{\lnbrk}}%
2412 \ifin@ \else \bbl@xin@{/k}{\bbl@cl{\lnbrk}} \fi
2413 \ifin@ \bbl@arabicjust \fi
2414 \bbl@xin@{/p}{\bbl@cl{\lnbrk}}%
2415 \ifin@ \AtBeginDocument{\@nameuse{\bbl@tibetanjust}} \fi
2416 % == Line breaking: hyphenate.other.(locale|script) ==
2417 \ifx\bbl@lbfkflag\@empty
2418   \bbl@ifunset{\bbl@hyotl@{\language\name}}{ }%
2419   {\bbl@csarg\bbl@replace{\hyotl@{\language\name}}{ }{,}%
2420   \bbl@startcommands*{\language\name}}%
2421   \bbl@csarg\bbl@foreach{\hyotl@{\language\name}}{ }%
2422   \ifcase\bbl@engine
2423     \ifnum##1<257
2424       \SetHyphenMap{\BabelLower{##1}{##1}}%
2425     \fi
2426   \else
2427     \SetHyphenMap{\BabelLower{##1}{##1}}%
2428   \fi}%
2429   \bbl@endcommands}%
2430 \bbl@ifunset{\bbl@hyots@{\language\name}}{ }%
2431 {\bbl@csarg\bbl@replace{\hyots@{\language\name}}{ }{,}%
2432 \bbl@csarg\bbl@foreach{\hyots@{\language\name}}{ }%
2433 \ifcase\bbl@engine
2434   \ifnum##1<257
2435     \global\lccode##1=##1\relax
2436   \fi
2437   \else
2438     \global\lccode##1=##1\relax
2439   \fi}}%
2440 \fi
2441 % == Counters: maparabic ==
2442 % Native digits, if provided in ini (TeX level, xe and lua)
2443 \ifcase\bbl@engine\else
2444   \bbl@ifunset{\bbl@dgnat@{\language\name}}{ }%
2445   {\expandafter\ifx\csname\bbl@dgnat@{\language\name}\endcsname\@empty\else
2446     \expandafter\expandafter\expandafter
2447     \bbl@setdigits\csname\bbl@dgnat@{\language\name}\endcsname
2448     \ifx\bbl@KVP@maparabic\@nnil\else
2449       \ifx\bbl@latinarabic\@undefined
2450         \expandafter\let\expandafter\@arabic
2451         \csname\bbl@counter@{\language\name}\endcsname
2452       \else % i.e., if layout=counters, which redefines \@arabic
2453         \expandafter\let\expandafter\bbl@latinarabic
2454         \csname\bbl@counter@{\language\name}\endcsname
2455       \fi
2456     \fi
2457   \fi}%
2458 \fi
2459 % == Counters: mapdigits ==
2460 % > luababel.def

```

```

2461 % == Counters: alph, Alph ==
2462 \ifx\bbbl@KVP@alph\@nnil\else
2463   \bbbl@exp{%
2464     \\bbbl@add\<bbbl@preextras@\language\name>{%
2465       \\babel@save\\@alph
2466       \let\\@alph\<bbbl@cntr@\bbbl@KVP@alph @\language\name>}}%
2467 \fi
2468 \ifx\bbbl@KVP@Alph\@nnil\else
2469   \bbbl@exp{%
2470     \\bbbl@add\<bbbl@preextras@\language\name>{%
2471       \\babel@save\\@Alph
2472       \let\\@Alph\<bbbl@cntr@\bbbl@KVP@Alph @\language\name>}}%
2473 \fi
2474 % == Counters: mapdot ==
2475 \ifx\bbbl@KVP@mapdot\@nnil\else
2476   \bbbl@foreach\bbbl@list@the{%
2477     \bbbl@ifunset{the##1}{}%
2478     {{\bbbl@ncarg\let\bbbl@tempd{the##1}%
2479       \bbbl@carg\bbbl@sreplace{the##1}{.}{\bbbl@map@lbl{.}}%
2480       \expandafter\ifx\csname the##1\endcsname\bbbl@tempd\else
2481         \bbbl@exp{\gdef\<the##1>{{\the##1}}}%
2482         \fi}}%
2483   \edef\bbbl@tempb{enumi,enumii,enumiii,enumiv}%
2484   \bbbl@foreach\bbbl@tempb{%
2485     \bbbl@ifunset{label##1}{}%
2486     {{\bbbl@ncarg\let\bbbl@tempd{label##1}%
2487       \bbbl@carg\bbbl@sreplace{label##1}{.}{\bbbl@map@lbl{.}}%
2488       \expandafter\ifx\csname label##1\endcsname\bbbl@tempd\else
2489         \bbbl@exp{\gdef\<label##1>{{\label##1}}}%
2490         \fi}}%
2491 \fi
2492 % == Casing ==
2493 \bbbl@release@casing
2494 \ifx\bbbl@KVP@casing\@nnil\else
2495   \bbbl@csarg\xdef{casing@\language\name}%
2496   {\@nameuse{bbbl@casing@\language\name}\bbbl@maybextx\bbbl@KVP@casing}%
2497 \fi
2498 % == Calendars ==
2499 \ifx\bbbl@KVP@calendar\@nnil
2500   \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}%
2501 \fi
2502 \def\bbbl@tempe##1 ##2\@{% Get first calendar
2503   \def\bbbl@tempa{##1}}%
2504   \bbbl@exp{\bbbl@tempe\bbbl@KVP@calendar\space\\@}%
2505 \def\bbbl@tempe##1.##2.##3.##4\@{%
2506   \def\bbbl@tempf{##3}% First
2507   \let\bbbl@tempc\@empty \let\bbbl@tempb\@empty
2508   \bbbl@cal@getconvert\bbbl@tempc{##1}%
2509   \bbbl@cal@getconvert\bbbl@tempb{##2}}%
2510 \expandafter\bbbl@tempe\bbbl@tempa...\@
2511 \bbbl@csarg\edef{calpr@\language\name}{%
2512   \ifx\bbbl@tempc\@empty\else
2513     calendar=\bbbl@tempc
2514   \fi
2515   \ifx\bbbl@tempb\@empty\else
2516     ,variant=\bbbl@tempb
2517   \fi
2518   \ifx\bbbl@tempf\@empty\else
2519     ,convert=\bbbl@tempf
2520   \fi}%
2521 % == engine specific extensions ==
2522 % Defined in XXXbabel.def
2523 \bbbl@provide@extra{##2}%

```

```

2524 % == require.babel in ini ==
2525 % To load or reload the babel-*.tex, if require.babel in ini
2526 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2527   \bbl@ifunset{bbl@rqtex@\language\name}{}%
2528     {\expandafter\ifx\csname bbl@rqtex@\language\name\endcsname\@empty\else
2529       \let\BabelBeforeIni@gobbletwo
2530       \chardef\atcatcode=\catcode\@
2531       \catcode\@=11\relax
2532       \def\CurrentOption{#2}%
2533       \bbl@input@texini{\bbl@cs{rqtex@\language\name}}%
2534       \catcode\@=\atcatcode
2535       \let\atcatcode\relax
2536       \global\bbl@csarg\let{rqtex@\language\name}\relax
2537     \fi}%
2538 \bbl@foreach\bbl@calendars{%
2539   \bbl@ifunset{bbl@ca##1}{%
2540     \chardef\atcatcode=\catcode\@
2541     \catcode\@=11\relax
2542     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2543     \catcode\@=\atcatcode
2544     \let\atcatcode\relax}%
2545   {}}%
2546 \fi
2547 % == frenchspacing ==
2548 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2549 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2550 \ifin@
2551   \bbl@extras@wrap{\bbl@pre@fs}%
2552   {\bbl@pre@fs}%
2553   {\bbl@post@fs}%
2554 \fi
2555 % == transforms ==
2556 % > luababel.def
2557 \def\CurrentOption{#2}%
2558 \@nameuse{bbl@icsave#2}%
2559 % == main ==
2560 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2561   \let\language\bbl@savelangname
2562   \chardef\localeid\bbl@savelocaleid\relax
2563 \fi
2564 % == ==
2565 \ifx\ldf@finish\@onlypreamble\else
2566   \bbl@afterload
2567 \fi
2568 % == hyphenrules (apply if current) ==
2569 \ifx\bbl@KVP@hyphenrules\@nnil\else
2570   \ifnum\bbl@savelocaleid=\localeid
2571     \language\@nameuse{l@\language\name}%
2572   \fi
2573 \fi}

```

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

```

2574 \def\bbl@provide@new#1{%
2575   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2576   \@namedef{extras#1}{}%
2577   \@namedef{noextras#1}{}%
2578   \bbl@startcommands*{#1}{captions}%
2579   \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2580     \def\bbl@tempb##1{% elt for \bbl@captionslist
2581       \if##1\@nnil\else
2582         \bbl@exp{%
2583           \SetString\##1%

```

```

2584         \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2585     \expandafter\bbl@tempb
2586 \fi}%
2587 \expandafter\bbl@tempb\bbl@captionslist\@nnil
2588 \else
2589     \ifx\bbl@initoload\relax
2590         \bbl@read@ini{\bbl@KVP@captions}2%    % Here letters cat = 11
2591     \else
2592         \bbl@read@ini{\bbl@initoload}2%        % Same
2593     \fi
2594 \fi
2595 \StartBabelCommands*{#1}{date}%
2596 \ifx\bbl@KVP@date\@nnil
2597     \bbl@exp{%
2598         \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
2599 \else
2600     \bbl@savetoday
2601     \bbl@savedate
2602 \fi
2603 \bbl@endcommands
2604 \bbl@load@basic{#1}%
2605 % == hyphenmins == (only if new)
2606 \bbl@exp{%
2607     \gdef\<#1hyphenmins>{%
2608         {\bbl@ifunset{\bbl@lfthm#1}{2}{\bbl@cs{lfthm#1}}}%
2609         {\bbl@ifunset{\bbl@rgthm#1}{3}{\bbl@cs{rgthm#1}}}}}%
2610 % == hyphenrules (also in renew) ==
2611 \bbl@provide@hyphens{#1}%
2612 % == main ==
2613 \ifx\bbl@KVP@main\@nnil\else
2614     \expandafter\main@language\expandafter{#1}%
2615 \fi}
2616 %
2617 \def\bbl@provide@renew#1{%
2618     \ifx\bbl@KVP@captions\@nnil\else
2619         \StartBabelCommands*{#1}{captions}%
2620         \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2621     \EndBabelCommands
2622 \fi
2623 \ifx\bbl@KVP@date\@nnil\else
2624     \StartBabelCommands*{#1}{date}%
2625     \bbl@savetoday
2626     \bbl@savedate
2627 \EndBabelCommands
2628 \fi
2629 % == hyphenrules (also in new) ==
2630 \ifx\bbl@lbfkflag\@empty
2631     \bbl@provide@hyphens{#1}%
2632 \fi
2633 % == main ==
2634 \ifx\bbl@KVP@main\@nnil\else
2635     \expandafter\main@language\expandafter{#1}%
2636 \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.

```

2637 \def\bbl@load@basic#1{%
2638     \ifcase\bbl@howloaded\or\or
2639         \ifcase\csname bbl@llevel@\language\endcsname
2640             \bbl@csarg\let{lname@\language}\relax
2641         \fi
2642     \fi

```

```

2643 \bbl@ifunset{bbl@lname@#1}%
2644 {\def\BabelBeforeIni##1##2{%
2645   \begingroup
2646     \let\bbl@ini@captions@aux\@gobbletwo
2647     \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2648     \bbl@read@ini{##1}l%
2649     \ifx\bbl@initoload\relax\endinput\fi
2650   \endgroup}%
2651   \begingroup      % boxed, to avoid extra spaces:
2652   \ifx\bbl@initoload\relax
2653     \bbl@input@texini{#1}%
2654   \else
2655     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2656   \fi
2657   \endgroup}%
2658   {}%

```

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

```

2659 \def\bbl@load@info#1{%
2660   \def\BabelBeforeIni##1##2{%
2661     \begingroup
2662       \bbl@read@ini{##1}0%
2663       \endinput      % babel- .tex may contain onlypreamble's
2664       \endgroup}%    boxed, to avoid extra spaces:
2665   {\bbl@input@texini{#1}}%

```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```

2666 \def\bbl@provide@hyphens#1{%
2667   \@tempcnta@m@ne % a flag
2668   \ifx\bbl@KVP@hyphenrules\@nnil\else
2669     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2670     \bbl@foreach\bbl@KVP@hyphenrules{%
2671       \ifnum\@tempcnta=\m@ne % if not yet found
2672         \bbl@ifsamestring{##1}{+}%
2673         {\bbl@carg\addlanguage{l@#1}}%
2674         {}%
2675         \bbl@ifunset{l@#1}% After a possible +
2676         {}%
2677         {\@tempcnta\@nameuse{l@#1}}%
2678       \fi}%
2679   \ifnum\@tempcnta=\m@ne
2680     \bbl@warning{%
2681       Requested 'hyphenrules' for '\language' not found:\\%
2682       \bbl@KVP@hyphenrules.\\%
2683       Using the default value. Reported}%
2684   \fi
2685   \fi
2686   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2687     \ifx\bbl@KVP@captions@\@nnil
2688       \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2689       {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2690        {}%
2691        {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2692        {}% if hyphenrules found:
2693        {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2694   \fi
2695   \fi
2696   \bbl@ifunset{l@#1}%
2697   {\ifnum\@tempcnta=\m@ne
2698     \bbl@carg\adddialect{l@#1}\language

```

```

2699 \else
2700 \bbl@carg\adddialect{l@#1}\@tempcnta
2701 \fi}%
2702 {\ifnum\@tempcnta=\m@ne\else
2703 \global\bbl@carg\chardef{l@#1}\@tempcnta
2704 \fi}}

```

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

```

2705 \def\bbl@input@texini#1{%
2706 \bbl@bsphack
2707 \bbl@exp{%
2708 \catcode`\\=\14 \catcode`\\=\0
2709 \catcode`\\=\1 \catcode`\\=\2
2710 \lowercase{\InputIfFileExists{babel-#1.tex}}{}}%
2711 \catcode`\\=\the\catcode`\relax
2712 \catcode`\\=\the\catcode`\\relax
2713 \catcode`\\=\the\catcode`\relax
2714 \catcode`\\=\the\catcode`\relax}%
2715 \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.

```

2716 \def\bbl@iniline#1\bbl@iniline{%
2717 \@ifnextchar[\bbl@iniset{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2718 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2719 \def\bbl@iniskip#1\@@{% if starts with ;
2720 \def\bbl@inistore#1=#2\@@{% full (default)
2721 \bbl@trim@def\bbl@tempa{#1}%
2722 \bbl@trim\toks@{#2}%
2723 \bbl@ifsamestring{\bbl@tempa}{\include}%
2724 {\bbl@read@subini{\the\toks@}}%
2725 {\bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2726 \ifin@ \else
2727 \bbl@xin@{,identification/include.}%
2728 {,\bbl@section/\bbl@tempa}%
2729 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2730 \bbl@exp{%
2731 \\g@addto@macro\\bbl@inidata{%
2732 \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2733 \fi}}
2734 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2735 \bbl@trim@def\bbl@tempa{#1}%
2736 \bbl@trim\toks@{#2}%
2737 \bbl@xin@{.identification.}{.\bbl@section.}%
2738 \ifin@
2739 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2740 \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2741 \fi}

```

4.19. Main loop in 'provide'

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

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

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

```

2742 \def\bbl@loop@ini#1{%
2743   \loop
2744     \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2745     \endlinechar\m@ne
2746     \read#1 to \bbl@line
2747     \endlinechar\^^M
2748     \ifx\bbl@line\empty\else
2749       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2750     \fi
2751   \repeat}
2752 %
2753 \def\bbl@read@subini#1{%
2754   \ifx\bbl@readsubstream\undefined
2755     \csname newread\endcsname\bbl@readsubstream
2756   \fi
2757   \openin\bbl@readsubstream=babel-#1.ini
2758   \ifeof\bbl@readsubstream
2759     \bbl@error{no-ini-file}{#1}{}{}%
2760   \else
2761     {\bbl@loop@ini\bbl@readsubstream}%
2762   \fi
2763   \closein\bbl@readsubstream}
2764 %
2765 \ifx\bbl@readstream\undefined
2766   \csname newread\endcsname\bbl@readstream
2767 \fi
2768 \def\bbl@read@ini#1#2{%
2769   \global\let\bbl@extend@ini@gobble
2770   \openin\bbl@readstream=babel-#1.ini
2771   \ifeof\bbl@readstream
2772     \bbl@error{no-ini-file}{#1}{}{}%
2773   \else
2774     % == Store ini data in \bbl@inidata ==
2775     \catcode\ =10 \catcode\ =12
2776     \catcode\[ =12 \catcode\] =12 \catcode\ =12 \catcode\& =12
2777     \catcode\; =12 \catcode\| =12 \catcode\% =14 \catcode\ - =12
2778     \ifnum#2=\m@ne % Just for the info
2779       \edef\language{tag \bbl@metalang}%
2780     \fi
2781     \ifnum#2=\m@ne
2782       \bbl@info{Metadata: '\bbl@metalang' requested, '#1' provided.\\%
2783         Fetching the locale name from babel-#1.ini\@gobble}%
2784     \else
2785       \bbl@info{Importing
2786         \ifcase#2font and identification \or basic \fi
2787         data for '\language'\%
2788         from babel-#1.ini. Reported}%
2789     \fi
2790     \ifnum#2<\@ne
2791       \global\let\bbl@inidata\empty
2792       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2793     \fi
2794     \def\bbl@section{identification}%
2795     \bbl@exp{%
2796       \\ \bbl@inistore tag.ini=#1\\ \@
2797       \\ \bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\ \@}%
2798     \bbl@loop@ini\bbl@readstream
2799     % == Process stored data ==
2800     \ifnum#2=\m@ne
2801       \def\bbl@tempa##1 ##2\@{##1}% Get first name
2802       \def\bbl@elt##1##2##3{%

```

```

2803     \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2804     {\edef\languagename{\bbl@tempa##3 \@@}%
2805     \let\localename\languagename
2806     \bbl@id@assign
2807     \def\bbl@elt###1###2###3{}}%
2808     {}}%
2809     \bbl@inidata
2810 \fi
2811 \bbl@csarg\xdef\luni@\languagename}{#1}%
2812 \bbl@read@ini@aux
2813 % == 'Export' data ==
2814 \bbl@ini@exports{#2}%
2815 \global\bbl@csarg\let\inidata@\languagename\bbl@inidata
2816 \global\let\bbl@inidata\empty
2817 \bbl@xin@{,\languagename,},{,\bbl@ini@loaded,}%
2818 \ifin@else
2819     \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
2820     \bbl@tglobal\bbl@ini@loaded
2821 \fi
2822 \@ifpackagewith{babel}{licr=unextended}{}%
2823     {\ifcase\bbl@engine\else % Find a better place
2824     \bbl@xin@{,\bbl@cl{sbcpr},{,Cyr1,}%
2825     \ifin@
2826     \bbl@once{licr-cryl}{\g@addto@macro\bbl@afterload{\input{cyr12uni.def}}}%
2827     \fi
2828     \fi}%
2829 \fi
2830 \closein\bbl@readstream}
2831 \def\bbl@read@ini@aux{%
2832     \let\bbl@savestrings\empty
2833     \let\bbl@savetoday\empty
2834     \let\bbl@savestate\empty
2835     \def\bbl@elt###1##2###3{%
2836     \def\bbl@section{##1}%
2837     \in@{=date.}{=##1}% Find a better place
2838     \ifin@
2839     \bbl@ifunset\bbl@inikv@##1{%
2840     {\bbl@ini@calendar{##1}}%
2841     {}}%
2842     \fi
2843     \bbl@ifunset\bbl@inikv@##1{}}%
2844     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2845 \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.

```

2846 \def\bbl@extend@ini@aux#1{%
2847     \bbl@startcommands*{#1}{captions}%
2848     % Activate captions/... and modify exports
2849     \bbl@csarg\def\inikv@captions.licr}##1##2{%
2850     \setlocalecaption{#1}{##1}{##2}}%
2851     \def\bbl@inikv@captions##1##2{%
2852     \bbl@ini@captions@aux{##1}{##2}}%
2853     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2854     \def\bbl@exportkey##1##2##3{%
2855     \bbl@ifunset\bbl@kv@##2{}}%
2856     {\expandafter\ifx\csname bbl@kv@##2\endcsname\empty\else
2857     \bbl@exp{\global\let<\bbl@##1@\languagename>\<\bbl@kv@##2>}}%
2858     \fi}}%
2859 % As with \bbl@read@ini, but with some changes
2860 \bbl@read@ini@aux
2861 \bbl@ini@exports\tw@
2862 % Update inidata@lang by pretending the ini is read.

```

```

2863 \def\bbl@elt##1##2##3{%
2864 \def\bbl@section{##1}%
2865 \bbl@iniline##2=##3\bbl@iniline}%
2866 \csname bbl@inidata@#1\endcsname
2867 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2868 \StartBabelCommands*{#1}{date}% And from the import stuff
2869 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2870 \bbl@savetoday
2871 \bbl@savestate
2872 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2873 \def\bbl@ini@calendar#1{%
2874 \lowercase{\def\bbl@tempa{=#1=}}%
2875 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2876 \bbl@replace\bbl@tempa{=date.}{}%
2877 \in@{.licr=}{#1=}%
2878 \ifin@
2879 \ifcase\bbl@engine
2880 \bbl@replace\bbl@tempa{.licr=}{}%
2881 \else
2882 \let\bbl@tempa\relax
2883 \fi
2884 \fi
2885 \ifx\bbl@tempa\relax\else
2886 \bbl@replace\bbl@tempa{=}{}%
2887 \ifx\bbl@tempa@empty\else
2888 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2889 \fi
2890 \bbl@exp{%
2891 \def\<bbl@inikv@#1>####1####2{%
2892 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2893 \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).

```

2894 \def\bbl@renewinikey#1/#2\@#3{%
2895 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2896 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2897 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2898 \bbl@trim\toks@{#3}% value
2899 \bbl@exp{%
2900 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2901 \\\g@addto@macro\\bbl@inidata{%
2902 \\\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.

```

2903 \def\bbl@exportkey#1#2#3{%
2904 \bbl@ifunset{\bbl@kv@#2}%
2905 {\bbl@csarg\gdef{#1@ \language name}{#3}}%
2906 {\expandafter\ifx\csname bbl@kv@#2\endcsname \@empty
2907 \bbl@csarg\gdef{#1@ \language name}{#3}}%
2908 \else
2909 \bbl@exp{\global\let\<bbl@#1@ \language name>\<bbl@kv@#2>}%
2910 \fi}}

```

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

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

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

```
2911 \def\bbl@iniwarning#1{%
2912   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2913   {\bbl@warning{%
2914     From babel-\bbl@cs{lini@language}.ini:\\%
2915     \bbl@cs{@kv@identification.warning#1}\\%
2916     Reported}}}%
2917 %
2918 \let\bbl@release@transforms\@empty
2919 \let\bbl@release@casing\@empty
```

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

```
2920 \def\bbl@ini@exports#1{%
2921   % Identification always exported
2922   \bbl@iniwarning{}%
2923   \ifcase\bbl@engine
2924     \bbl@iniwarning{.pdfLaTeX}%
2925   \or
2926     \bbl@iniwarning{.luaLaTeX}%
2927   \or
2928     \bbl@iniwarning{.XeLaTeX}%
2929   \fi%
2930   \bbl@exportkey{lllevel}{identification.load.level}{}%
2931   \bbl@exportkey{elname}{identification.name.english}{}%
2932   \bbl@expf{\bbl@exportkey{lname}{identification.name.opentype}%
2933     {\csname bbl@elname@language\endcsname}}%
2934   \bbl@exportkey{tbcpr}{identification.tag.bcp47}{}%
2935   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2936   \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
2937   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2938   \bbl@exportkey{esname}{identification.script.name}{}%
2939   \bbl@expf{\bbl@exportkey{sname}{identification.script.name.opentype}%
2940     {\csname bbl@esname@language\endcsname}}%
2941   \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2942   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2943   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2944   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2945   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2946   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2947   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2948   % Also maps bcp47 -> language
2949   \bbl@csarg\def{bcp@map@bbl@cl{tbcpr}}{\language}%
2950   \ifcase\bbl@engine\or
2951     \directlua{%
2952       Babel.locale_props[\the\bbl@cs{id@language}].script
2953       = '\bbl@cl{sbcpr}'}%
2954   \fi
2955   % Conditional
2956   \ifnum#1>\z@ % -1 or 0 = only info, 1 = basic, 2 = (re)new
2957     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2958     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2959     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2960     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
```

```

2961 \bbl@exportkey{rgthm}{typography.righthypenmin}{3}%
2962 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2963 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2964 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2965 \bbl@exportkey{intsp}{typography.intraspace}{}%
2966 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2967 \bbl@exportkey{chrng}{characters.ranges}{}%
2968 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2969 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2970 \ifnum#1=\tw@ % only (re)new
2971 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2972 \bbl@tglobal\bbl@savetoday
2973 \bbl@tglobal\bbl@savedate
2974 \bbl@savestrings
2975 \fi
2976 \fi}

```

4.20. Processing keys in ini

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

```

2977 \def\bbl@inikv#1#2{%      key=value
2978 \toks@{#2}%              This hides #'s from ini values
2979 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2980 \let\bbl@inikv@identification\bbl@inikv
2981 \let\bbl@inikv@date\bbl@inikv
2982 \let\bbl@inikv@typography\bbl@inikv
2983 \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.

```

2984 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\empty x-\fi}
2985 \def\bbl@inikv@characters#1#2{%
2986 \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2987 {\bbl@exp{%
2988 \\\g@addto@macro{\bbl@release@casing{%
2989 \\\bbl@casemapping}{\language}\unexpanded{#2}}}%
2990 {\in@{casing.}{#1}% e.g., casing.Uv = uV
2991 \ifin@
2992 \lowercase{\def\bbl@tempb{#1}%
2993 \bbl@replace\bbl@tempb{casing.}}}%
2994 \bbl@exp{\\\g@addto@macro{\bbl@release@casing{%
2995 \\\bbl@casemapping
2996 {\\\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2997 \else
2998 \bbl@inikv{#1}{#2}%
2999 \fi}}

```

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

```

3000 \def\bbl@inikv@counters#1#2{%
3001 \bbl@ifsamestring{#1}{digits}%
3002 {\bbl@error{digits-is-reserved}}{}{}%
3003 {}%
3004 \def\bbl@tempc{#1}%
3005 \bbl@trim@def{\bbl@tempb*}{#2}%
3006 \in@{.1$}{#1$}%
3007 \ifin@
3008 \bbl@replace\bbl@tempc{.1}{}%
3009 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%

```

```

3010 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3011 \fi
3012 \in@{.F.}{#1}%
3013 \ifin@else\in@{.S.}{#1}\fi
3014 \ifin@
3015 \bbl@csarg\protected@xdef{cnt@#1@language}{\bbl@tempb*}%
3016 \else
3017 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3018 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3019 \bbl@csarg\global\expandafter\let{cnt@#1@language}\bbl@tempa
3020 \fi
3021 \in@{.D$}{#1$}%
3022 \ifin@else\in@{.C$}{#1$}\fi
3023 \ifin@
3024 \bbl@replace\bbl@tempc{.D}{}%
3025 \bbl@replace\bbl@tempc{.C}{}%
3026 \bbl@exp\gdef\<bbl@cnt@bbl@tempc @language>####1{%
3027 \ \expandafter\ \bbl@zhnumeral\ \expandafter{\ \number####1}{\bbl@tempc}}}%
3028 \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.

```

3029 \ifcase\bbl@engine
3030 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3031 \bbl@ini@captions@aux{#1}{#2}}
3032 \else
3033 \def\bbl@inikv@captions#1#2{%
3034 \bbl@ini@captions@aux{#1}{#2}}
3035 \fi

```

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

```

3036 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3037 \bbl@replace\bbl@tempa{.template}{}%
3038 \def\bbl@toreplace{#1}{}%
3039 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3040 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3041 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3042 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}}%
3043 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}}%
3044 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3045 \ifin@
3046 \@nameuse{\bbl@patch\bbl@tempa}%
3047 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3048 \fi
3049 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3050 \ifin@
3051 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3052 \bbl@exp\gdef\<fnum@\bbl@tempa>{%
3053 \ \bbl@ifunset{\bbl@bbl@tempa fmt@\ \language}%
3054 {[fnum@\bbl@tempa]}%
3055 {\ \@nameuse{\bbl@bbl@tempa fmt@\ \language}}}%
3056 \fi}
3057 %
3058 \def\bbl@ini@captions@aux#1#2{%
3059 \bbl@trim\def\bbl@tempa{#1}%
3060 \bbl@xin@{.template}{\bbl@tempa}%
3061 \ifin@
3062 \bbl@ini@captions@template{#2}\language
3063 \else
3064 \bbl@ifblank{#2}%
3065 {\bbl@exp{%
3066 \toks@{\ \bbl@nocaption{\bbl@tempa name}{\language\bbl@tempa name}}}%
3067 {\bbl@trim\toks@{#2}}}%

```

```

3068 \bbl@exp{%
3069   \\\bbl@add\\bbl@savestrings{%
3070     \\\SetString<\bbl@tempa name>{\the\toks@}}}%
3071 \toks@ \expandafter{\bbl@captionslist}%
3072 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}}%
3073 \ifin@ \else
3074   \bbl@exp{%
3075     \\\bbl@add<\bbl@extracaps@ \language name>{\<\bbl@tempa name>}}%
3076     \\\bbl@to global \<\bbl@extracaps@ \language name>}%
3077 \fi
3078 \fi}

```

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

```

3079 \def\bbl@list@the{%
3080   part,chapter,section,subsection,subsubsection,paragraph,%
3081   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3082   table,page,footnote,mpfootnote,mpfn}
3083 %
3084 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3085   \bbl@ifunset{\bbl@map@#1@ \language name}%
3086     {\@nameuse{#1}}}%
3087     {\@nameuse{\bbl@map@#1@ \language name}}}%
3088 %
3089 \def\bbl@map@lbl#1{% #1:a sign, eg, .
3090   \ifin@ \else
3091     \bbl@ifunset{\bbl@map@#1@ \language name}%
3092       {#1}%
3093       {\@nameuse{\bbl@map@#1@ \language name}}}%
3094 \fi}
3095 %
3096 \def\bbl@inikv@labels#1#2{%
3097   \in@{.map}{#1}%
3098   \ifin@
3099     \in@{,dot.map,}{, #1,}%
3100   \ifin@
3101     \global\@namedef{\bbl@map@. @ \language name}{#2}%
3102   \fi
3103   \ifx\bbl@KVP@labels\@nnil\else
3104     \bbl@xin@{ map }{ \bbl@KVP@labels \space}%
3105     \ifin@
3106       \def\bbl@tempc{#1}%
3107       \bbl@replace\bbl@tempc{.map}{}%
3108       \in@{, #2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3109       \bbl@exp{%
3110         \gdef<\bbl@map@ \bbl@tempc @ \language name>%
3111         {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3112       \bbl@foreach\bbl@list@the{%
3113         \bbl@ifunset{the##1}{}%
3114         {\bbl@ncarg\let\bbl@tempd{the##1}%
3115          \bbl@exp{%
3116            \\\bbl@sreplace<the##1>%
3117            {\<\bbl@tempc>{##1}}%
3118            {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3119            \\\bbl@sreplace<the##1>%
3120            {\<\@empty @ \bbl@tempc>\<c@##1>%
3121             {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3122             \\\bbl@sreplace<the##1>%
3123             {\\\csname @ \bbl@tempc \\\endcsname \<c@##1>%
3124              {\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3125             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3126               \bbl@exp{\gdef<the##1>{\[the##1]}}%
3127             \fi}}%
3128       \fi

```

```

3129 \fi
3130 %
3131 \else
3132 % The following code is still under study. You can test it and make
3133 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3134 % language dependent.
3135 \in@{enumerate.}{#1}%
3136 \ifin@
3137 \def\bbl@tempa{#1}%
3138 \bbl@replace\bbl@tempa{enumerate.}{}%
3139 \def\bbl@toreplace{#2}%
3140 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3141 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3142 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3143 \toks@{\expandafter\bbl@toreplace}%
3144 \bbl@exp{%
3145 \\\bbl@add<extras\language>{%
3146 \\\babel@save<labelenum\romannumeral\bbl@tempa>%
3147 \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3148 \\\bbl@tglobal<extras\language>}%
3149 \fi
3150 \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.

```

3151 \def\bbl@chapttype{chapter}
3152 \ifx\@makechapterhead\@undefined
3153 \let\bbl@patchchapter\relax
3154 \else\ifx\thechapter\@undefined
3155 \let\bbl@patchchapter\relax
3156 \else\ifx\ps@headings\@undefined
3157 \let\bbl@patchchapter\relax
3158 \else
3159 \def\bbl@patchchapter{%
3160 \global\let\bbl@patchchapter\relax
3161 \gdef\bbl@chfmt{%
3162 \bbl@ifunset\bbl@chapttype fmt@\language}%
3163 {\@chapapp\space\thechapter}%
3164 {\@nameuse\bbl@chapttype fmt@\language}}%
3165 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3166 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3167 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3168 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3169 \bbl@tglobal\appendix
3170 \bbl@tglobal\ps@headings
3171 \bbl@tglobal\chaptermark
3172 \bbl@tglobal\@makechapterhead}
3173 \let\bbl@patchappendix\bbl@patchchapter
3174 \fi\fi\fi
3175 \ifx\@part\@undefined
3176 \let\bbl@patchpart\relax
3177 \else
3178 \def\bbl@patchpart{%
3179 \global\let\bbl@patchpart\relax
3180 \gdef\bbl@partformat{%
3181 \bbl@ifunset\bbl@partfmt@\language}%
3182 {\@partname\nobreakspace\thepart}%
3183 {\@nameuse\bbl@partfmt@\language}}%
3184 \bbl@sreplace\@part{\@partname\nobreakspace\thepart}{\bbl@partformat}%
3185 \bbl@tglobal\@part}
3186 \fi

```

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

```

3187 \def\bbl@cal@getconvert#1#2{% Used in provide
3188   \in{:}{#2}%
3189   \ifin@def\bbl@tempf{#2}\else\def#1{#2}\fi
3190 \let\bbl@calendar@empty
3191 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3192 \def\bbl@localedate#1#2#3#4{%
3193   \begingroup
3194     \edef\bbl@they{#2}%
3195     \edef\bbl@them{#3}%
3196     \edef\bbl@thed{#4}%
3197     \edef\bbl@tempe{%
3198       \bbl@ifunset\bbl@calpr{\language\language}\bbl@cl{calpr}},%
3199       #1}%
3200     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3201     \let\bbl@ld@calendar@empty
3202     \let\bbl@ld@variant@empty
3203     \let\bbl@ld@convert@relax
3204     \let\bbl@ld@@convert@relax
3205     \expandafter\bbl@forkv\expandafter{\bbl@tempe}{\bbl@csarg\edef{ld##1}{##2}}%
3206     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3207     \ifx\bbl@ld@convert@relax
3208       \let\bbl@ld@convert\bbl@ld@@convert
3209     \fi
3210     \ifx\bbl@ld@convert@empty
3211       \ifx\bbl@ld@calendar@empty
3212         \let\bbl@ld@convert@relax
3213       \else
3214         \let\bbl@ld@convert\bbl@ld@calendar
3215       \fi
3216     \fi
3217     \ifx\bbl@ld@convert@relax\else
3218       \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3219       {\bbl@ld@convert}\bbl@they\bbl@them\bbl@thed
3220     \fi
3221     \@nameuse{\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3222     \edef\bbl@calendar{% Used in \month..., too
3223       \bbl@ld@calendar
3224       \ifx\bbl@ld@variant@empty\else
3225         .\bbl@ld@variant
3226       \fi}%
3227     \bbl@cased
3228     {\@nameuse{\bbl@date@\language @\bbl@calendar}%
3229      \bbl@they\bbl@them\bbl@thed}%
3230   \endgroup}
3231 %
3232 \def\bbl@printdate#1{%
3233   \ifnextchar[\bbl@printdate@i{#1}]{\bbl@printdate@i{#1}[]}}
3234 \def\bbl@printdate@i#1[#2]#3#4#5{%
3235   \bbl@usedategrouptrue
3236   \def\bbl@tempc{#1}%
3237   \bbl@replace\bbl@tempc{-}{@}%
3238   \@nameuse{\bbl@ensure@\bbl@tempc}{\localedate[#2][#3][#4][#5]}
3239 %
3240 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3241 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3242   \bbl@trim@def\bbl@tempa{#1.#2}%
3243   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3244   {\bbl@trim@def\bbl@tempa{#3}%
3245    \bbl@trim\toks@{#5}%
3246    \@temptokena\expandafter{\bbl@savedate}%
3247    \bbl@exp{% Reverse order - in ini last wins

```

```

3248 \def\\bbl@savestate{%
3249   \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3250   \the\temptokena}}}%
3251 {\bbl@ifsamestring{\bbl@tempa}{date.long}%    defined now
3252 {\lowercase{\def\bbl@tempb{#6}}}%
3253 \bbl@trim@def\bbl@toreplace{#5}%
3254 \bbl@TG@@date
3255 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3256 \ifx\bbl@savestate\empty
3257 \bbl@exp{%
3258   \\AfterBabelCommands{%
3259     \gdef\<\language date>{\\protect\<\language date >}%
3260     \gdef\<\language date >{\\bbl@printdate{\language}}}%
3261   \def\\bbl@savestate{%
3262     \\SetString\\today{%
3263       \<\language date>[@convert]%
3264       {\the\year}{\the\month}{\the\day}}}%
3265   \fi}%
3266 {}}}

```

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).

```

3267 \let\bbl@calendar\empty
3268 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3269   \bbl@xin@{\$calendrica:}{\$#2}%
3270   \ifin@
3271     \directlua{Babel.calendrica = Babel.calendrica or require("calendrica")}%
3272     \edef\bbl@tempa{\$#2}%
3273     \bbl@replace\bbl@tempa{\$calendrica:}{}%
3274     \bbl@afterelse
3275     \bbl@exp{\\bbl@calendrica#1\\@@{\bbl@tempa}}%
3276   \else
3277     \bbl@afterfi
3278     \@nameuse{bbl@ca@#2}#1\@@
3279   \fi}
3280 \def\bbl@calendrica#1-#2-#3\@@#4#5#6#7{%
3281   \directlua{
3282     local cldc = require(calendrica).cal_gregorian
3283     if cldc then
3284       local d = Babel.calendrica.new_date(
3285         {year=number#1, month=number#2, day=number#3},
3286         Babel.calendrica.cal_gregorian)
3287       local r = Babel.calendrica.as_date(d, Babel.calendrica.cal_#4)
3288       token.set_macro('\bbl@stripslash#5', r.year)
3289       token.set_macro('\bbl@stripslash#6', r.month)
3290       token.set_macro('\bbl@stripslash#7', r.day)
3291     else
3292       local r = Babel.calendrica.date(
3293         {year=number#1, month=number#2, day=number#3},
3294         'gregorian'):to('#4')
3295       token.set_macro('\bbl@stripslash#5', tostring(r.year))
3296       token.set_macro('\bbl@stripslash#6', tostring(r.month))
3297       token.set_macro('\bbl@stripslash#7', tostring(r.day))
3298     end}}
3299 \newcommand\babelDateSpace{\nobreakspace}
3300 \newcommand\babelDateDot{.\@}
3301 \newcommand\babelDated[1][{\number#1}]
3302 \newcommand\babelDatedd[1][{\ifnum#1<10 0\fi\number#1}]
3303 \newcommand\babelDateM[1][{\number#1}]
3304 \newcommand\babelDateMM[1][{\ifnum#1<10 0\fi\number#1}]

```

```

3305 \newcommand\BabelDateMMMM[1]{\%
3306 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3307 \newcommand\BabelDatey[1]{\number#1}%
3308 \newcommand\BabelDateyy[1]{\%
3309 \ifnum#1<10 0\number#1 %
3310 \else\ifnum#1<100 \number#1 %
3311 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3312 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3313 \else
3314 \bbl@error{limit-two-digits}{\}\}%
3315 \fi\fi\fi\fi}}
3316 \newcommand\BabelDateyyyy[1]{\number#1}}
3317 \newcommand\BabelDateU[1]{\number#1}%
3318 \def\bbl@replace@finish@iii#1{%
3319 \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3320 \def\bbl@TG@date{%
3321 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3322 \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot{}}%
3323 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3324 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[###1]}%
3325 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3326 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3327 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3328 \bbl@replace\bbl@toreplace{[M]}{\bbl@datecctr[###2]}%
3329 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3330 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3331 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3332 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[###3]}%
3333 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3334 \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3335 \bbl@replace\bbl@toreplace{[U]}{\bbl@datecctr[###1]}%
3336 \bbl@replace@finish@iii\bbl@toreplace}
3337 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3338 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

4.21. French spacing (again)

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

```

3339 \AddToHook{begindocument/before}{%
3340 \let\bbl@normalsf\normalsfcodes
3341 \let\normalsfcodes\relax}
3342 \AtBeginDocument{%
3343 \ifx\bbl@normalsf\@empty
3344 \ifnum\sfcodes\@m
3345 \let\normalsfcodes\frenchspacing
3346 \else
3347 \let\normalsfcodes\nonfrenchspacing
3348 \fi
3349 \else
3350 \let\normalsfcodes\bbl@normalsf
3351 \fi}

```

Transforms.

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

```

3352 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3353 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3354 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3355 #1[#2]{#3}{#4}{#5}}

```

```

3356 \begingroup
3357 \catcode`\%=12
3358 \catcode`\&=14
3359 \gdef\bbl@transforms#1#2#3{%&
3360 \directlua{
3361     local str = [==[#2]==]
3362     str = str:gsub('%.%d+.%d+$', '')
3363     token.set_macro('babeltempa', str)
3364 }&%
3365 \def\babeltempc{%&
3366 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}%&
3367 \ifin@else
3368 \bbl@xin@{:,\babeltempa,}{,\bbl@KVP@transforms,}%&
3369 \fi
3370 \ifin@
3371 \bbl@foreach\bbl@KVP@transforms{%&
3372 \bbl@xin@{:,\babeltempa,}{,##1,}%&
3373 \ifin@ &% font:font:transform syntax
3374 \directlua{
3375     local t = {}
3376     for m in string.gmatch('##1'..' ':'', '(.):') do
3377         table.insert(t, m)
3378     end
3379     table.remove(t)
3380     token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3381 }&%
3382 \fi}%&
3383 \in@{.0$}{#2$}%&
3384 \ifin@
3385 \directlua{%& (\attribute) syntax
3386     local str = string.match([[ \bbl@KVP@transforms]],
3387         '%([^(%[-])%)[^)]-\babeltempa')
3388     if str == nil then
3389         token.set_macro('babeltempb', '')
3390     else
3391         token.set_macro('babeltempb', ',attribute=' .. str)
3392     end
3393 }&%
3394 \toks@{#3}%&
3395 \bbl@exp{%&
3396 \\\g@addto@macro\\\bbl@release@transforms{%&
3397 \relax &% Closes previous \bbl@transforms@aux
3398 \\\bbl@transforms@aux
3399 \\\#1{label=\babeltempa\babeltempb\babeltempc}%&
3400 {\language\the\toks@}}}%&
3401 \else
3402 \g@addto@macro\bbl@release@transforms{, {#3}}}%&
3403 \fi
3404 \fi}
3405 \endgroup

```

4.22. Handle language system

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

```

3406 \def\bbl@provide@lsys#1{%
3407 \bbl@ifunset\bbl@lname@#1}%
3408 {\bbl@load@info@#1}%
3409 }%
3410 \bbl@csarg\let{lsys@#1}\@empty
3411 \bbl@ifunset\bbl@sname@#1{\bbl@csarg\gdef{sname@#1}{Default}}}%

```

```

3412 \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef\sotf@#1{\DFLT}}{}%
3413 \bbl@csarg\bbl@add@list{\sys@#1}{Script=\bbl@cs{sname@#1}}%
3414 \bbl@ifunset{\bbl@lname@#1}{}%
3415 {\bbl@csarg\bbl@add@list{\sys@#1}{Language=\bbl@cs{lname@#1}}}%
3416 \ifcase\bbl@engine\or\or
3417 \bbl@ifunset{\bbl@prehc@#1}{}%
3418 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3419 {}%
3420 {\ifx\bbl@xenohyph\undefined
3421 \global\let\bbl@xenohyph\bbl@xenohyph@d
3422 \ifx\AtBeginDocument\@notprerr
3423 \expandafter\@secondoftwo % to execute right now
3424 \fi
3425 \AtBeginDocument{%
3426 \bbl@patchfont{\bbl@xenohyph}%
3427 {\expandafter\select@language\expandafter{\language}}}%
3428 \fi}}%
3429 \fi
3430 \bbl@csarg\bbl@toglobal{\sys@#1}

```

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

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

```

3467 \else
3468 \toks@{\expandafter{\the\toks@}\or #1}%
3469 \expandafter\bbl@buildifcase
3470 \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 a special case, for a fixed form (see babel-he.ini, for example).

```

3471 \newcommand\localenumeral[2]{%
3472 \bbl@ifunset{\bbl@cntr@#1@\language}%
3473 {#2}%
3474 {\bbl@cs{cntr@#1@\language}{#2}}}
3475 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3476 \newcommand\localecounter[2]{%
3477 \expandafter\bbl@localecntr
3478 \expandafter{\number\csname c@#2\endcsname}{#1}}
3479 \def\bbl@alphnumeral#1#2{%
3480 \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3481 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3482 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3483 \bbl@alphnumeral@ii{#9}000000#1\or
3484 \bbl@alphnumeral@ii{#9}00000#1#2\or
3485 \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3486 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3487 \bbl@error{counter-too-large}{>9999}{}}%
3488 \fi}
3489 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3490 \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3491 {\bbl@cs{cntr@#1.4@\language}#5%
3492 \bbl@cs{cntr@#1.3@\language}#6%
3493 \bbl@cs{cntr@#1.2@\language}#7%
3494 \bbl@cs{cntr@#1.1@\language}#8%
3495 \ifnum#6#7#8>\z@
3496 \bbl@ifunset{\bbl@cntr@#1.S.321@\language}{}}%
3497 {\bbl@cs{cntr@#1.S.321@\language}}%
3498 \fi}%
3499 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}

```

Some Chinese counters follow special rules.

```

3500 \def\bbl@zhnumeral#1#2{%
3501 \ifnum#1<10 \bbl@zhnum000#1{#2}%
3502 \else\ifnum#1<100 \bbl@zhnum00#1{#2}%
3503 \else\ifnum#1<1000 \bbl@zhnum0#1{#2}%
3504 \else\ifnum#1<10000 \bbl@zhnum#1{#2}%
3505 \else\bbl@error{counter-too-large}{#1}{}}%
3506 \fi\fi\fi\fi}
3507 \def\bbl@zhnum#1#2#3#4#5{%
3508 \ifnum\numexpr#1#2#3#4\relax=\z@
3509 \localenumeral{#5.C}\@ne
3510 \else
3511 \ifnum#1>\z@\localenumeral{#5.D}{#1}\localenumeral{#5.C}5\fi % Thousands
3512 \ifnum#2>\z@\localenumeral{#5.D}{#2}\localenumeral{#5.C}4\fi % Hundreds
3513 \else\ifnum#1>\z@\ifnum#3#4>\z@\localenumeral{#5.C}\@ne\fi
3514 \fi\fi
3515 \ifnum#3>\z@ % Tens
3516 \ifnum#1#2=\z@\ifnum#3=\@ne\else\localenumeral{#5.D}{#3}\fi % No 千 for 10-19:
3517 \else\localenumeral{#5.D}{#3}%
3518 \fi
3519 \localenumeral{#5.C}\thr@@
3520 \else
3521 \ifnum#2>\z@\ifnum#4>\z@\localenumeral{#5.C}\@ne\fi\fi
3522 \fi

```

```

3523 \ifnum#4>\z@\localenumeral{#5.D}{#4}\fi % Units
3524 \fi}

```

4.24. Casing

```

3525 \newcommand\BabelUppercaseMapping[3]{%
3526 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3527 \newcommand\BabelTitlecaseMapping[3]{%
3528 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3529 \newcommand\BabelLowercaseMapping[3]{%
3530 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

The parser for casing and casing. (variant).
3531 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3532 \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3533 \else
3534 \def\bbl@uftocode#1{\expandafter`\string#1}
3535 \fi
3536 \def\bbl@casemapping#1#2#3{% 1:variant
3537 \def\bbl@tempa##1 ##2{% Loop
3538 \bbl@casemapping@i{##1}%
3539 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3540 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3541 \def\bbl@tempe{0}% Mode (upper/lower...)
3542 \def\bbl@tempc{#3}% Casing list
3543 \expandafter\bbl@tempa\bbl@tempc\@empty}
3544 \def\bbl@casemapping@i#1{%
3545 \def\bbl@tempb{#1}%
3546 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3547 \@nameuse{regex_replace_all:nnN}%
3548 {[{\x{c0}-\x{ff}}][\x{80}-\x{bf}]}*}{\0}}\bbl@tempb
3549 \else
3550 \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb
3551 \fi
3552 \expandafter\bbl@casemapping@ii\bbl@tempb\@
3553 \def\bbl@casemapping@ii#1#2#3\@{%
3554 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3555 \ifin@
3556 \edef\bbl@tempe{%
3557 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3558 \else
3559 \ifcase\bbl@tempe\relax
3560 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3561 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3562 \or
3563 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3564 \or
3565 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3566 \or
3567 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3568 \fi
3569 \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.

```

3570 \def\bbl@localeinfo#1#2{%
3571 \bbl@ifunset{bbl@info@#2}{#1}%
3572 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3573 {\bbl@cs{csname bbl@info@#2\endcsname @\languagename}}}}
3574 \newcommand\localeinfo[1]{%
3575 \ifx*#1\@empty
3576 \bbl@afterelse\bbl@localeinfo{}%

```

```

3577 \else
3578 \bbl@localeinfo
3579 {\bbl@error{no-ini-info}{}}{}%
3580 {#1}%
3581 \fi}
3582 % \@namedef{bbl@info@name.locale}{lcname}
3583 \@namedef{bbl@info@tag.ini}{lini}
3584 \@namedef{bbl@info@name.english}{elname}
3585 \@namedef{bbl@info@name.opentype}{lname}
3586 \@namedef{bbl@info@tag.bcp47}{tbc}
3587 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3588 \@namedef{bbl@info@tag.opentype}{lotf}
3589 \@namedef{bbl@info@script.name}{esname}
3590 \@namedef{bbl@info@script.name.opentype}{sname}
3591 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3592 \@namedef{bbl@info@script.tag.opentype}{sotf}
3593 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3594 \@namedef{bbl@info@variant.tag.bcp47}{vbc}
3595 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3596 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3597 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}

```

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

```

3598 (< *More package options) >=
3599 \DeclareOption{ensureinfo=off}{}
3600 (< /More package options) >
3601 \let\BabelEnsureInfo\relax

```

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

```

3602 \newcommand\getlocaleproperty{%
3603 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3604 \def\bbl@getproperty@s#1#2#3{%
3605 \let#1\relax
3606 \def\bbl@elt##1##2##3{%
3607 \bbl@ifsamestring{##1/##2}{#3}%
3608 {\providecommand#1{##3}%
3609 \def\bbl@elt###1####2####3{}}}%
3610 {}}%
3611 \bbl@cs{inidata@#2}}%
3612 \def\bbl@getproperty@x#1#2#3{%
3613 \bbl@getproperty@s{#1}{#2}{#3}%
3614 \ifx#1\relax
3615 \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3616 \fi}

```

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

```

3617 \let\bbl@ini@loaded\@empty
3618 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3619 \def\ShowLocaleProperties#1{%
3620 \typeout{}%
3621 \typeout{*** Properties for language '#1' ***}%
3622 \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3623 \@nameuse{bbl@inidata@#1}%
3624 \typeout{*****}}

```

4.26. BCP 47 related commands

This macro is called by language selectors when the language isn't recognized. So, it's the core for (1) mapping from a BCP 47 tag to the actual language, if `bcp47.toname` is enabled (i.e., if `\bbl@bcp47toname` is true), and (2) lazy loading. With `\autoload.bcp47` enabled *and* lazy loading, we must first build a name for the language, with the help of `\autoload.bcp47.prefix`. Then we use `\provideprovide` passing the options set with `\autoload.bcp47.options` (by default import). Finally, and if the locale

has not been loaded before, we use `\provideprovide` with the language name as passed to the selector.

```

3625 \newif\ifbbl@bcpallowed
3626 \bbl@bcpallowedfalse
3627 \def\bbl@autoload@options{@import}
3628 \def\bbl@provide@locale{%
3629   \ifx\babelprovide\undefined
3630     \bbl@error{base-on-the-fly}{}}}%
3631 \fi
3632 \let\bbl@auxname\language
3633 \ifbbl@bcptoname
3634   \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
3635   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
3636   \let\locale\language}%
3637 \fi
3638 \ifbbl@bcpallowed
3639   \expandafter\ifx\csname date\language\endcsname\relax
3640     \expandafter
3641     \bbl@bcplookup{\language}%
3642     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3643       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3644       \let\locale\language
3645       \expandafter\ifx\csname date\language\endcsname\relax
3646         \let\bbl@initoload\bbl@bcp
3647         \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3648         \let\bbl@initoload\relax
3649       \fi
3650       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\locale}%
3651     \fi
3652   \fi
3653 \fi
3654 \expandafter\ifx\csname date\language\endcsname\relax
3655   \IfFileExists{babel-\language.tex}%
3656   {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3657   {}%
3658 \fi}

```

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

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

```

3659 \providecommand\BCPdata{}
3660 \ifx\renewcommand\undefined\else
3661   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3662   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3663     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3664     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3665     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3666   \def\bbl@bcpdata@ii#1#2{%
3667     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3668     {\bbl@error{unknown-ini-field}{#1}}}%
3669     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3670     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}
3671 \fi
3672 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3673 \@namedef{bbl@info@tag.tag.bcp47}{tbcpl} % For \BCPdata

```

5. Adjusting the Babel behavior

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

```

3674 \newcommand\babeladjust[1]{%

```

```

3675 \bbl@forkv{#1}{%
3676 \bbl@ifunset{bbl@ADJ@##1@##2}%
3677 {\bbl@cs{ADJ@##1}{##2}}%
3678 {\bbl@cs{ADJ@##1@##2}}}%
3679 %
3680 \def\bbl@adjust@lua#1#2{%
3681 \ifvmode
3682 \ifnum\currentgrouplevel=\z@
3683 \directlua{ Babel.#2 }%
3684 \expandafter\expandafter\expandafter\@gobble
3685 \fi
3686 \fi
3687 {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3688 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3689 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3690 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3691 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3692 \@namedef{bbl@ADJ@bidi.text@on}{%
3693 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3694 \@namedef{bbl@ADJ@bidi.text@off}{%
3695 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3696 \@namedef{bbl@ADJ@bidi.math@on}{%
3697 \let\bbl@noamsmath\@empty}
3698 \@namedef{bbl@ADJ@bidi.math@off}{%
3699 \let\bbl@noamsmath\relax}
3700 %
3701 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3702 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3703 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3704 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3705 %
3706 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3707 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3708 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3709 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3710 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3711 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3712 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3713 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3714 \@namedef{bbl@ADJ@justify.arabic@on}{%
3715 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3716 \@namedef{bbl@ADJ@justify.arabic@off}{%
3717 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3718 %
3719 \def\bbl@adjust@layout#1{%
3720 \ifvmode
3721 #1%
3722 \expandafter\@gobble
3723 \fi
3724 {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3725 \@namedef{bbl@ADJ@layout.tabular@on}{%
3726 \ifnum\bbl@tabular@mode=\tw@
3727 \bbl@adjust@layout{\let\@tabular\bbl@NL@\@tabular}%
3728 \else
3729 \chardef\bbl@tabular@mode\@ne
3730 \fi}
3731 \@namedef{bbl@ADJ@layout.tabular@off}{%
3732 \ifnum\bbl@tabular@mode=\tw@
3733 \bbl@adjust@layout{\let\@tabular\bbl@OL@\@tabular}%
3734 \else
3735 \chardef\bbl@tabular@mode\z@
3736 \fi}
3737 \@namedef{bbl@ADJ@layout.lists@on}{%

```

```

3738 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3739 \@namedef{bbl@ADJ@layout.lists@off}{%
3740 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3741 %
3742 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3743 \bbl@bcppallowedtrue}
3744 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3745 \bbl@bcppallowedfalse}
3746 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3747 \def\bbl@bcp@prefix{#1}}
3748 \def\bbl@bcp@prefix{bcp47-}
3749 \@namedef{bbl@ADJ@autoload.options}#1{%
3750 \def\bbl@autoload@options{#1}}
3751 \def\bbl@autoload@bcptoptions{import}
3752 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3753 \def\bbl@autoload@bcptoptions{#1}}
3754 \newif\ifbbl@bcptname
3755 %
3756 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3757 \bbl@bcptonametrue}
3758 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3759 \bbl@bcptonamefalse}
3760 %
3761 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3762 \directlua{ Babel.ignore_pre_char = function(node)
3763     return (node.lang == \the\csname l@nohyphenation\endcsname)
3764 end }}
3765 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3766 \directlua{ Babel.ignore_pre_char = function(node)
3767     return false
3768 end }}
3769 %
3770 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3771 \def\bbl@ignoreinterchar{%
3772 \ifnum\language=\l@nohyphenation
3773 \expandafter\@gobble
3774 \else
3775 \expandafter\@firstofone
3776 \fi}}
3777 \@namedef{bbl@ADJ@interchar.disable@off}{%
3778 \let\bbl@ignoreinterchar\@firstofone}
3779 %
3780 \@namedef{bbl@ADJ@select.write@shift}{%
3781 \let\bbl@restorelastskip\relax
3782 \def\bbl@savelastskip{%
3783 \let\bbl@restorelastskip\relax
3784 \ifvmode
3785 \ifdim\lastskip=\z@
3786 \let\bbl@restorelastskip\nobreak
3787 \else
3788 \bbl@exp{%
3789 \def\\bbl@restorelastskip{%
3790 \skip@=\the\lastskip
3791 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3792 \fi
3793 \fi}}
3794 \@namedef{bbl@ADJ@select.write@keep}{%
3795 \let\bbl@restorelastskip\relax
3796 \let\bbl@savelastskip\relax}
3797 \@namedef{bbl@ADJ@select.write@omit}{%
3798 \AddBabelHook{babel-select}{beforestart}{%
3799 \expandafter\babel@aux\expandafter\bbl@main@language{}}}%
3800 \let\bbl@restorelastskip\relax

```

```

3801 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3802 \@namedef{bbl@ADJ@select.encoding@off}{%
3803 \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.

```

3804 ({*More package options}) ≡
3805 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3806 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3807 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3808 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3809 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3810 ({/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.

```

3811 \bbl@trace{Cross referencing macros}
3812 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3813 \def\@newl@bel#1#2#3{%
3814 {\@safe@activestrue
3815 \bbl@ifunset{#1@#2}%
3816 \relax
3817 {\gdef\@multiplelabels{%
3818 \latex@warning@no@line{There were multiply-defined labels}}%
3819 \latex@warning@no@line{Label `#2' multiply defined}}%
3820 \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.

```

3821 \CheckCommand*\@testdef[3]{%
3822 \def\reserved@a{#3}%
3823 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3824 \else
3825 \@tempswatrue
3826 \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.

```

3827 \def\@testdef#1#2#3{%
3828 \@safe@activestrue
3829 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3830 \def\bbl@tempb{#3}%
3831 \@safe@activesfalse
3832 \ifx\bbl@tempa\relax
3833 \else
3834 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3835 \fi
3836 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%

```

```

3837 \ifx\bbl@tempa\bbl@tempb
3838 \else
3839 \@tempswattrue
3840 \fi}
3841 \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.

```

3842 \bbl@xin@{R}\bbl@opt@safe
3843 \ifin@
3844 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3845 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3846 {\expandafter\strip@prefix\meaning\ref}%
3847 \ifin@
3848 \bbl@redefine\@kernel@ref#1{%
3849 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3850 \bbl@redefine\@kernel@pageref#1{%
3851 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3852 \bbl@redefine\@kernel@sref#1{%
3853 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3854 \bbl@redefine\@kernel@spageref#1{%
3855 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3856 \else
3857 \bbl@redefineroobust\ref#1{%
3858 \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3859 \bbl@redefineroobust\pageref#1{%
3860 \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3861 \fi
3862 \else
3863 \let\org@ref\ref
3864 \let\org@pageref\pageref
3865 \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.

```

3866 \bbl@xin@{B}\bbl@opt@safe
3867 \ifin@
3868 \bbl@redefine\@citex[#1]#2{%
3869 \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetrue}
3870 \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.)

```

3871 \AtBeginDocument{%
3872 \ifpackageloaded{natbib}{%
3873 \def\@citex[#1][#2]#3{%
3874 \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activetrue}
3875 \org@@citex[#1][#2]{\bbl@tempa}}%
3876 }{}%

```

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

```

3877 \AtBeginDocument{%
3878   \@ifpackageloaded{cite}{%
3879     \def\@citex[#1]#2{%
3880       \@safe@activestrue\org@citex[#1]{#2}\@safe@activesfalse}%
3881     }{}}

```

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

```

3882 \bbl@redefine\nocite#1{%
3883   \@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.

```

3884 \bbl@redefine\bibcite{%
3885   \bbl@cite@choice
3886   \bibcite}

```

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

```

3887 \def\bbl@bibcite#1#2{%
3888   \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.

```

3889 \def\bbl@cite@choice{%
3890   \global\let\bibcite\bbl@bibcite
3891   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3892     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3893       \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.

```

3894 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3895 \bbl@redefine\@bibitem#1{%
3896   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3897 \else
3898   \let\org@nocite\nocite
3899   \let\org@citex\@citex
3900   \let\org@bibcite\bibcite
3901   \let\org@bibitem\@bibitem
3902 \fi

```

5.2. Layout

```

3903 \newcommand\BabelPatchSection[1]{%
3904   \@ifundefined{#1}{}{%
3905     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3906     \@namedef{#1}{%
3907       \ifstar{\bbl@presec@s{#1}}%
3908       {\dblarg{\bbl@presec@x{#1}}}}}%
3909 \def\bbl@presec@x#1[#2]#3{%
3910   \bbl@exp{%

```

```

3911 \\select@language@x{\bbl@main@language}%
3912 \\bbl@cs{sspre@#1}%
3913 \\bbl@cs{ss@#1}%
3914 [\\foreignlanguage{\language}{\unexpanded{#2}}]%
3915 {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3916 \ifcase\bbl@bidimode\else\\babelsublr{}\fi}%
3917 \\select@language@x{\language}%
3918 \def\bbl@presec@s#1#2{%
3919 \bbl@exp{%
3920 \\select@language@x{\bbl@main@language}%
3921 \\bbl@cs{sspre@#1}%
3922 \\bbl@cs{ss@#1}*%
3923 {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3924 \ifcase\bbl@bidimode\else\\babelsublr{}\fi}%
3925 \\select@language@x{\language}%
3926 %
3927 \IfBabelLayout{sectioning}%
3928 {\BabelPatchSection{part}%
3929 \BabelPatchSection{chapter}%
3930 \BabelPatchSection{section}%
3931 \BabelPatchSection{subsection}%
3932 \BabelPatchSection{subsubsection}%
3933 \BabelPatchSection{paragraph}%
3934 \BabelPatchSection{subparagraph}%
3935 \def\babel@toc#1{%
3936 \select@language@x{\bbl@main@language}}}%
3937 \IfBabelLayout{captions}%
3938 {\BabelPatchSection{caption}}}%

```

\BabelFootnote Footnotes.

```

3939 \bbl@trace{Footnotes}
3940 \def\bbl@footnote#1#2#3{%
3941 \@ifnextchar[%
3942 {\bbl@footnote@o{#1}{#2}{#3}}%
3943 {\bbl@footnote@x{#1}{#2}{#3}}}
3944 \long\def\bbl@footnote@x#1#2#3#4{%
3945 \bgroup
3946 \select@language@x{\bbl@main@language}%
3947 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3948 \egroup}
3949 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3950 \bgroup
3951 \select@language@x{\bbl@main@language}%
3952 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3953 \egroup}
3954 \def\bbl@footnotetext#1#2#3{%
3955 \@ifnextchar[%
3956 {\bbl@footnotetext@o{#1}{#2}{#3}}%
3957 {\bbl@footnotetext@x{#1}{#2}{#3}}}
3958 \long\def\bbl@footnotetext@x#1#2#3#4{%
3959 \bgroup
3960 \select@language@x{\bbl@main@language}%
3961 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3962 \egroup}
3963 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3964 \bgroup
3965 \select@language@x{\bbl@main@language}%
3966 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3967 \egroup}
3968 \def\BabelFootnote#1#2#3#4{%
3969 \ifx\bbl@fn@footnote\undefined
3970 \let\bbl@fn@footnote\footnote
3971 \fi

```

```

3972 \ifx\bbl@fn@footnotetext\@undefined
3973 \let\bbl@fn@footnotetext\footnotetext
3974 \fi
3975 \bbl@ifblank{#2}%
3976 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3977 \namedef{\bbl@stripslash#1text}%
3978 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3979 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3980 \namedef{\bbl@stripslash#1text}%
3981 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3982 \IfBabelLayout{footnotes}%
3983 {\let\bbl@OL@footnote\footnote
3984 \BabelFootnote\footnote\language{}{}}%
3985 \BabelFootnote\localfootnote\language{}{}}%
3986 \BabelFootnote\mainfootnote{}{}{}}
3987 {}

```

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.

```

3988 \bbl@trace{Marks}
3989 \IfBabelLayout{sectioning}
3990 {\ifx\bbl@opt@headfoot\@nnil
3991 \g@addto@macro\@resetactivechars{%
3992 \set@typeset@protect
3993 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3994 \let\protect\noexpand
3995 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3996 \edef\thepage{%
3997 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3998 \fi}%
3999 \fi}
4000 {\ifbbl@single\else
4001 \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
4002 \markright#1{%
4003 \bbl@ifblank{#1}%
4004 {\org@markright{}}}%
4005 {\toks@{#1}%
4006 \bbl@exp{%
4007 \org@markright{\protect\foreignlanguage{\language}%
4008 {\protect\bbl@restore@actives\the\toks@}}}}}%

```

\markboth

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

```

4009 \ifx\@mkboth\markboth
4010 \def\bbl@tempc{\let\@mkboth\markboth}%
4011 \else
4012 \def\bbl@tempc{}%
4013 \fi
4014 \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
4015 \markboth#1#2{%
4016 \protected@edef\bbl@tempb##1{%

```

```

4017         \protect\foreignlanguage
4018         {\language\name}{\protect\bbl@restore@actives##1}}%
4019     \bbl@ifblank{#1}%
4020     {\toks@{}}%
4021     {\toks@\expandafter{\bbl@tempb{#1}}}%
4022     \bbl@ifblank{#2}%
4023     {\@temptokena{}}%
4024     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
4025     \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
4026     \bbl@tempc
4027 \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.

```

4028 \bbl@trace{Preventing clashes with other packages}
4029 \ifx\org@ref\undefined\else
4030   \bbl@xin@{R}\bbl@opt@safe
4031   \ifin@
4032     \AtBeginDocument{%
4033       \@ifpackageloaded{ifthen}{%
4034         \bbl@redefine@long\ifthenelse#1#2#3{%
4035           \let\bbl@temp@pref\pageref
4036           \let\pageref\org@pageref
4037           \let\bbl@temp@ref\ref
4038           \let\ref\org@ref
4039           \@safe@activestrue
4040           \org@ifthenelse{#1}%
4041             {\let\pageref\bbl@temp@pref
4042              \let\ref\bbl@temp@ref
4043              \@safe@activesfalse
4044              #2}%
4045             {\let\pageref\bbl@temp@pref
4046              \let\ref\bbl@temp@ref
4047              \@safe@activesfalse
4048              #3}%
4049           }%
4050         }{}%
4051       }
4052 \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`.

```

4053 \AtBeginDocument{%
4054   \ifpackageloaded{varioref}{%
4055     \bbl@redefine\@@vpageref#1[#2]#3{%
4056       \@safe@activestrue
4057       \org@@vpageref{#1}[#2]{#3}%
4058       \@safe@activesfalse}%
4059     \bbl@redefine\vrefpagenum#1#2{%
4060       \@safe@activestrue
4061       \org@vrefpagenum{#1}{#2}%
4062       \@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.

```

4063   \expandafter\def\csname Ref \endcsname#1{%
4064     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa
4065     }{}%
4066   }
4067 \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.

```

4068 \AtEndOfPackage{%
4069   \AtBeginDocument{%
4070     \@ifpackageloaded{hhline}%
4071     {\expandafter\ifx\csname normal@char\string\endcsname\relax
4072       \else
4073         \makeatletter
4074         \def\@currname{hhline}\input{hhline.sty}\makeatother
4075         \fi}%
4076     {}%

```

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

```

4077 \def\substitutefontfamily#1#2#3{%
4078   \lowercase{\immediate\openout15=#1#2.fd\relax}%
4079   \immediate\write15{%
4080     \string\ProvidesFile{#1#2.fd}%
4081     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
4082     \space generated font description file]^J
4083     \string\DeclareFontFamily{#1}{#2}{^^J
4084     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
4085     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
4086     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
4087     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
4088     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
4089     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
4090     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
4091     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
4092     }%
4093   \closeout15

```

```

4094 }
4095 \@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

```

4096 \bbl@trace{Encoding and fonts}
4097 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
4098 \newcommand\BabelNonText{TS1,T3,TS3}
4099 \let\org@TeX\TeX
4100 \let\org@LaTeX\LaTeX
4101 \let\ensureascii\@firstofone
4102 \let\asciientcoding\@empty
4103 \AtBeginDocument{%
4104   \def\elt#1{,#1,%
4105   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4106   \let\elt\relax
4107   \let\bbl@tempb\@empty
4108   \def\bbl@tempc{OT1}%
4109   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
4110     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
4111   \bbl@foreach\bbl@tempa{%
4112     \bbl@xin@{,#1}{,\BabelNonASCII,%
4113     \ifin@
4114       \def\bbl@tempb{#1}% Store last non-ascii
4115     \else\bbl@xin@{,#1}{,\BabelNonText,% Pass
4116     \ifin@else
4117       \def\bbl@tempc{#1}% Store last ascii
4118     \fi
4119     \fi}%
4120   \ifx\bbl@tempb\@empty\else
4121     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,%
4122     \ifin@else
4123       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
4124     \fi
4125     \let\asciientcoding\bbl@tempc
4126     \renewcommand\ensureascii[1]{%
4127       {\fontencoding{\asciientcoding}\selectfont#1}}%
4128     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
4129     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
4130   \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.

```

4131 \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.

```

4132 \AtBeginDocument{%
4133   \@ifpackageloaded{fontspec}%
4134   {\xdef\latinencoding{%

```

```

4135     \ifx\UTFencname\undefined
4136       EU\ifcase\bbl@engine\or2\or1\fi
4137     \else
4138       \UTFencname
4139     \fi}}%
4140 {\gdef\latinencoding{OT1}%
4141  \ifx\cf@encoding\bbl@t@one
4142    \xdef\latinencoding{\bbl@t@one}%
4143  \else
4144    \def\@elt#1{,#1,}%
4145    \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4146    \let\@elt\relax
4147    \bbl@xin@{,T1,}\bbl@tempa
4148    \ifin@
4149      \xdef\latinencoding{\bbl@t@one}%
4150    \fi
4151  \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.

```

4152 \DeclareRobustCommand{\latintext}{%
4153   \fontencoding{\latinencoding}\selectfont
4154   \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.

```

4155 \ifx\@undefined\DeclareTextFontCommand
4156   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
4157 \else
4158   \DeclareTextFontCommand{\textlatin}{\latintext}
4159 \fi

```

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

```

4160 \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 `Lua $\TeX$ -ja` shows, vertical typesetting is possible, too.

```

4161 \bbl@trace{Loading basic (internal) bidi support}
4162 \ifodd\bbl@engine
4163 \else % Any xe+lua bidi

```

```

4164 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
4165   \bbl@error{bidi-only-lua}{}}{}{}%
4166   \let\bbl@beforeforeign\leavevmode
4167   \AtEndOfPackage{%
4168     \EnableBabelHook{babel-bidi}%
4169     \bbl@xebidipar}
4170 \fi\fi
4171 \def\bbl@loadxebidi#1{%
4172   \ifx\RTLfootnotetext\undefined
4173     \AtEndOfPackage{%
4174       \EnableBabelHook{babel-bidi}%
4175       \ifx\fontspec\undefined
4176         \usepackage{fontspec}% bidi needs fontspec
4177       \fi
4178       \usepackage#1{bidi}%
4179       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4180       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4181         \ifnum\@nameuse\bbl@wdir\language=\tw@ % 'AL' bidi
4182           \bbl@digitsdotdash % So ignore in 'R' bidi
4183         \fi}}%
4184   \fi}
4185 \ifnum\bbl@bidimode>200 % Any xe bidi=
4186   \ifcase\expandafter\gobbletwo\the\bbl@bidimode\or
4187     \bbl@tentative{bidi=bidi}
4188     \bbl@loadxebidi{}
4189   \or
4190     \bbl@loadxebidi{[rldocument]}
4191   \or
4192     \bbl@loadxebidi{}
4193   \fi
4194 \fi
4195 \fi
4196 \ifnum\bbl@bidimode=\@ne % bidi=default
4197   \let\bbl@beforeforeign\leavevmode
4198   \ifodd\bbl@engine % lua
4199     \newattribute\bbl@attr@dir
4200     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4201     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4202   \fi
4203   \AtEndOfPackage{%
4204     \EnableBabelHook{babel-bidi}% pdf/lua/xe
4205     \ifodd\bbl@engine\else % pdf/xe
4206       \bbl@xebidipar
4207     \fi}
4208 \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.

```

4209 \bbl@trace{Macros to switch the text direction}
4210 \def\bbl@alscripts{%
4211   ,Arabic,Syriac,Thaana,Hanifi_Rohingya,Hanifi,Sogdian,}
4212 \def\bbl@rscripts{%
4213   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4214   Hatran,Hebrew,Imperial_Aramaic,Inscriptional_Pahlavi,%
4215   Inscriptional_Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
4216   Mende_Kikakui,Meroitic_Cursive,Meroitic_Hieroglyphs,Nabataean,%
4217   Nko,Old_Hungarian,Old_North_Arabian,Old_Sogdian,%
4218   Old_South_Arabian,Old_Turkic,Old_Uyghur,Palmyrene,Phoenician,%
4219   Psalter_Pahlavi,Samaritan,Yezidi,Mandaean,%
4220   Meroitic,N'Ko,Orkhon,Todhri}
4221 %
4222 \def\bbl@provide@dirs#1{%

```

```

4223 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4224 \ifin@
4225 \global\bbl@csarg\chardef{wdir@#1}\@ne
4226 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4227 \ifin@
4228 \global\bbl@csarg\chardef{wdir@#1}\tw@
4229 \fi
4230 \else
4231 \global\bbl@csarg\chardef{wdir@#1}\z@
4232 \fi
4233 \ifodd\bbl@engine
4234 \bbl@csarg\ifcase{wdir@#1}%
4235 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4236 \or
4237 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4238 \or
4239 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4240 \fi
4241 \fi}
4242 %
4243 \def\bbl@switchdir{%
4244 \bbl@ifunset{\bbl@sys@\language}\bbl@provide@sys{\language}}{}%
4245 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
4246 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
4247 \def\bbl@setdirs#1{%
4248 \ifcase\bbl@select@type
4249 \bbl@bodydir{#1}%
4250 \bbl@pardir{#1}% <- Must precede \bbl@texdir
4251 \fi
4252 \bbl@texdir{#1}}
4253 \ifnum\bbl@bidimode>\z@
4254 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4255 \DisableBabelHook{babel-bidi}
4256 \fi

```

Now the engine-dependent macros.

```

4257 \ifodd\bbl@engine % luatex=1
4258 \else % pdftex=0, xetex=2
4259 \newcount\bbl@dirlevel
4260 \chardef\bbl@thetexdir\z@
4261 \chardef\bbl@thepardir\z@
4262 \def\bbl@texdir#1{%
4263 \ifcase#1\relax
4264 \chardef\bbl@thetexdir\z@
4265 \@nameuse{setlatin}%
4266 \bbl@texdir@i\beginL\endL
4267 \else
4268 \chardef\bbl@thetexdir\@ne
4269 \@nameuse{setnonlatin}%
4270 \bbl@texdir@i\beginR\endR
4271 \fi}
4272 \def\bbl@texdir@i#1#2{%
4273 \ifhmode
4274 \ifnum\currentgrouplevel>\z@
4275 \ifnum\currentgrouplevel=\bbl@dirlevel
4276 \bbl@error{multiple-bidi}{}}{}%
4277 \bgroup\aftergroup#2\aftergroup\egroup
4278 \else
4279 \ifcase\currentgrouptype\or % 0 bottom
4280 \aftergroup#2% 1 simple {}
4281 \or
4282 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4283 \or

```

```

4284      \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4285      \or\or\or % vbox vtop align
4286      \or
4287      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4288      \or\or\or\or\or\or % output math disc insert vcent mathchoice
4289      \or
4290      \aftergroup#2% 14 \beginngroup
4291      \else
4292      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4293      \fi
4294      \fi
4295      \bbl@dirlevel\currentgrouplevel
4296      \fi
4297      #1%
4298      \fi}
4299      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4300      \let\bbl@bodydir\@gobble
4301      \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).

```

4302      \def\bbl@xebidipar{%
4303      \let\bbl@xebidipar\relax
4304      \TeXeTstate\@ne
4305      \def\bbl@xeeverypar{%
4306      \ifcase\bbl@thepardir
4307      \ifcase\bbl@thetextdir\else\beginR\fi
4308      \else
4309      {\setbox\z@\lastbox\beginR\box\z@}%
4310      \fi}%
4311      \AddToHook{para/begin}{\bbl@xeeverypar}}
4312      \ifnum\bbl@bidimode>200 % Any xe bidi=
4313      \let\bbl@textdir\i\@gobbletwo
4314      \let\bbl@xebidipar\@empty
4315      \AddBabelHook{bidi}{foreign}{%
4316      \ifcase\bbl@thetextdir
4317      \BabelWrapText{\LR{##1}}}%
4318      \else
4319      \BabelWrapText{\RL{##1}}}%
4320      \fi}
4321      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4322      \fi
4323      \fi

```

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

```

4324      \providecommand\bbl@sublr[1]{\bbl@textdir\z@#1}
4325      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@sublr{##1}}}
4326      \AtBeginDocument{%
4327      \ifx\pdfstringdefDisableCommands\undefined\else
4328      \ifx\pdfstringdefDisableCommands\relax\else
4329      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4330      \fi
4331      \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`.

```

4332 \bbl@trace{Local Language Configuration}
4333 \ifx\loadlocalcfg\@undefined
4334   \ifpackagewith{babel}{noconfigs}%
4335     {\let\loadlocalcfg\@gobble}%
4336     {\def\loadlocalcfg#1{%
4337       \InputIfFileExists{#1.cfg}%
4338       {\typeout{*****^J%
4339                * Local config file #1.cfg used^^J%
4340                *}}}%
4341       \@empty}}
4342 \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).

```

4343 \bbl@trace{Language options}
4344 \def\BabelDefinitionFile#1#2#3{}
4345 \let\bbl@afterlang\relax
4346 \let\BabelModifiers\relax
4347 \let\bbl@loaded\@empty
4348 \def\bbl@load@language#1{%
4349   \InputIfFileExists{#1.ldf}%
4350   {\edef\bbl@loaded{CurrentOption
4351     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4352     \expandafter\let\expandafter\bbl@afterlang
4353       \csname\CurrentOption.ldf-h@k\endcsname
4354     \expandafter\let\expandafter\BabelModifiers
4355       \csname bbl@mod@\CurrentOption\endcsname
4356     \bbl@exp{\AtBeginDocument{%
4357       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4358     {\bbl@error{unknown-package-option}{}}{}}

```

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

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

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

```

4359 \ifx\GetDocumentProperties\@undefined\else
4360   \let\bbl@beforeforeign\leavevmode
4361   \edef\bbl@tempa{\GetDocumentProperties{document/other-languages}}%
4362   \let\bbl@collect@other\@empty
4363   \bbl@foreach\bbl@tempa{%
4364     \edef\bbl@metalang{#1}%
4365     \ifx\bbl@metalang\@empty\else
4366       \begingroup
4367         \expandafter
4368         \bbl@bcpllookup{\bbl@metalang}%
4369         \ifx\bbl@bcp\relax
4370           \ifx\bbl@opt@main\@nnil
4371             \bbl@error{no-locale-for-meta}{\bbl@metalang}{}{}%
4372           \fi
4373         \else
4374           \bbl@read@ini{\bbl@bcp}\m@ne

```

```

4375         \xdef\bbl@collect@other{\bbl@collect@other,\language}%
4376         \GenericInfo{}\{(babel) \@spaces\@spaces\@spaces
4377         Passing '\language' to babel\@gobble}%
4378     \fi
4379 \endgroup
4380 \fi}
4381 \ifx\bbl@collect@other\@empty\else
4382     \xdef\bbl@language@opts{\bbl@collect@other,\bbl@language@opts}%
4383 \fi
4384 \edef\bbl@metalang{\GetDocumentProperties{document/language}}%
4385 \ifx\bbl@metalang\@empty\else
4386     \begingroup
4387         \expandafter
4388         \bbl@bcplookup{\bbl@metalang}%
4389         \ifx\bbl@bcp\relax
4390             \ifx\bbl@opt@main\@nnil
4391                 \bbl@error{no-locale-for-meta}{\bbl@metalang}{\}%
4392             \fi
4393         \else
4394             \bbl@read@ini{\bbl@bcp}\m@ne
4395             \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4396             \ifx\bbl@opt@main\@nnil
4397                 \global\let\bbl@opt@main\language
4398             \fi
4399             \GenericInfo{}\{(babel) \@spaces\@spaces\@spaces
4400             Passing '\language' to babel\@gobble}%
4401         \fi
4402     \endgroup
4403 \fi
4404 \fi
4405 \ifx\bbl@opt@config\@nnil
4406     \@ifpackagewith{babel}{noconfigs}{\}%
4407     {\InputIfFileExists{bblopts.cfg}%
4408     {\bbl@info{Configuration files are deprecated, as\\%
4409         they can break document portability.\\%
4410         Reported}%
4411     \typeout{*****^J%
4412         * Local config file bblopts.cfg used^^J%
4413         *}}%
4414     }{\}%
4415 \else
4416     \InputIfFileExists{\bbl@opt@config.cfg}%
4417     {\bbl@info{Configuration files are deprecated, as\\%
4418         they can break document portability.\\%
4419         Reported}%
4420     \typeout{*****^J%
4421         * Local config file \bbl@opt@config.cfg used^^J%
4422         *}}%
4423     {\bbl@error{config-not-found}{\}{\}}%
4424 \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `\bbl@language@opts` are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the available locales, and which version (ldf or ini) will be loaded. This is done by first loading the corresponding `babel-⟨name⟩.tex` file.

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

loaded directly (with @import).

```

4425 \def\BabelBeforeIni#1#2{%
4426   \def\bbl@tempa{\@m}% <- Default if no \BDefFile
4427   \let\bbl@tempb\@empty
4428   #2%
4429   \edef\bbl@toload{%
4430     \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4431     \bbl@toload@last}%
4432   \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4433 \def\BabelDefinitionFile#1#2#3{%
4434   \def\bbl@tempa{#1}\def\bbl@tempb{#2}%
4435   \@namedef{bbl@preldf@CurrentOption}{#3}%
4436   \endinput}%

```

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

```

4437 \def\bbl@tempf{,}
4438 \@nameuse{clist_map_inline:Nn}\@raw@classoptionslist{%
4439   \in@{=}{#1}%
4440   \ifin@else
4441     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4442   \fi}

```

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

```

4443 \let\bbl@toload\@empty
4444 \let\bbl@toload@last\@empty
4445 \let\bbl@unkopt\@gobble %% <- Ugly
4446 \edef\bbl@tempc{%
4447   \bbl@tempf,@@,\bbl@language@opts
4448   \ifx\bbl@opt@main\@nnil\else,\bbl@opt@main\fi}
4449 \let\BabelLocalesTentative\bbl@tempc
4450 %
4451 \bbl@foreach\bbl@tempc{%
4452   \in@{@@}{#1}% <- Ugly
4453   \ifin@
4454     \def\bbl@unkopt##1{%
4455       \DeclareOption{##1}{\bbl@error{unknown-package-option}}{}}}%
4456   \else
4457     \def\CurrentOption{#1}%
4458     \bbl@xin@{/#1//}{\bbl@toload@last}% Collapse consecutive
4459     \ifin@else
4460       \lowercase{\InputIfFileExists{babel-#1.tex}}{ }{%
4461         \IfFileExists{#1.ldf}%
4462         {\edef\bbl@toload{%
4463           \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4464           \bbl@toload@last}%
4465           \edef\bbl@toload@last{0/0//\CurrentOption//und/#1}}%
4466         {\bbl@unkopt{#1}}}%
4467     \fi
4468   \fi}

```

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

```

4469 \ifx\bbl@opt@main\@nnil
4470   \ifx\bbl@toload@last\@empty
4471     \def\bbl@toload@last{0/0//nil//und-x-nil/nil}
4472     \bbl@info{%

```

```

4473     You haven't specified a language as a class or package\\%
4474     option. I'll load 'nil'. Reported}
4475 \fi
4476 \else
4477 \let\bbl@tempc\@empty
4478 \bbl@foreach\bbl@toload{%
4479 \bbl@xin@{//\bbl@opt@main//}{#1}%
4480 \ifin@else
4481 \bbl@add@list\bbl@tempc{#1}%
4482 \fi}
4483 \let\bbl@toload\bbl@tempc
4484 \fi
4485 \edef\bbl@toload{\bbl@toload,1\bbl@toload@last}

```

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

```

4486 \def\AfterBabelLanguage#1{%
4487 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4488 \NewHook{babel/presets}
4489 \UseHook{babel/presets}
4490 %
4491 \let\bbl@tempb\@empty
4492 \def\bbl@tempc#1/#2/#3/#4/#5\@@{%
4493 \count@z@
4494 \ifnum#2=\@m % if no \BabelDefinitionFile
4495 \ifnum#1=z@ % not main. -- % if provide+=!, provide*=!
4496 \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0/#3/#4/#3\@@
4497 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4498 \fi
4499 \else % 10 = main -- % if provide+=!, provide*=!
4500 \ifodd\bbl@ldfflag\bbl@tempc 10/0/#3/#4/#3\@@
4501 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4502 \fi
4503 \fi
4504 \else
4505 \ifnum#1=z@ % not main
4506 \ifnum\bbl@iniflag>\@ne\else % if 0, provide
4507 \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4508 \fi
4509 \else % 10 = main
4510 \ifodd\bbl@iniflag\else % if provide+, provide*
4511 \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4512 \fi
4513 \fi
4514 \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4515 \fi}

```

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

```

4516 \def\bbl@tempd#1#2#3#4#5{%
4517 \DeclareOption{#3}{%
4518 \ifcase\count@
4519 \bbl@exp{\bbl@add\bbl@tempb{%
4520 \global\let\bbl@afterload\@empty
4521 \\\@nameuse\bbl@preini{#3}%
4522 \bbl@ldffinit
4523 \def\CurrentOption{#3}%
4524 \\\babelprovide[import=#4,\ifnum#1=z@\else\bbl@opt@provide,main\fi]{#3}%
4525 \bbl@afterldf
4526 \bbl@afterload}}%
4527 \else
4528 \bbl@add\bbl@tempb{%

```

```

4529 \global\let\bbl@afterload\@empty
4530 \def\CurrentOption{#3}%
4531 \let\localename\CurrentOption
4532 \let\language\localename
4533 \def\BabelIniTag{#4}%
4534 \@nameuse{bbl@preldf@#3}%
4535 \begingroup
4536 \bbl@id@assign
4537 \bbl@read@ini{\BabelIniTag}0%
4538 \endgroup
4539 \bbl@load@language{#5}%
4540 \bbl@afterload}%
4541 \fi}
4542 %
4543 \bbl@foreach\bbl@toload{\bbl@tempc#1\@@}
4544 \bbl@tempb
4545 \DeclareOption*{}
4546 \ProcessOptions
4547 %
4548 \bbl@exp{%
4549 \\\AtBeginDocument{\bbl@usehooks@lang{/}{\begindocument}}{}}}%
4550 \def\AfterBabelLanguage{\bbl@error{late-after-babel}}{}{}{}
4551 \AddToHook{begindocument/end}{%
4552 \bbl@info{Main language set to '\mainlocalename'\@gobble}}
4553 {/package}

```

6. Tagging

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

```

4554 {*tagging}
4555 \ifx\bbl@chfmt\@undefined\else
4556 \EditInstance{heading}{chapter}{%
4557 prefix = ,
4558 number-format = \unskip\bbl@chfmt
4559 }
4560 \fi
4561 \ifx\bbl@partformat\@undefined\else
4562 \EditInstance{heading}{part}{%
4563 prefix = ,
4564 number-format = \unskip\bbl@partformat
4565 }
4566 \fi
4567 \@ifpackagewith{babel}{babel-metadata=tagging}{\endinput}
4568 \let\bbl@ifmetadata\IfDocumentMetadataT
4569 \ExplSyntaxOn
4570 \NewTaggingSocket{babel/foreign/begin}{0}
4571 \NewTaggingSocket{babel/foreign/end}{0}
4572 \NewTaggingSocketPlug{babel/foreign/begin}{default}{
4573 \tag_mc_end_push:
4574 % \bbl@array@get\bbl@select@opts{raw}\bbl@tempa
4575 % \ifx\bbl@tempa\relax\let\bbl@tempa\@empty\fi
4576 \bbl@array@get\bbl@select@opts{actualtext}\bbl@tempb
4577 \ifx\bbl@tempb\relax\let\bbl@tempb\@empty\fi
4578 \tagstructbegin{tag=Span, lang=\BCPDdata{tag},
4579 % raw=\bbl@tempa,
4580 actualtext=\bbl@tempb}
4581 \tagmcbegin {} }
4582 \NewTaggingSocketPlug{babel/foreign/end}{default}{
4583 \tagmcend
4584 \tagstructend

```

```

4585 \tag_mc_begin_pop:n {}
4586 \AssignTaggingSocketPlug{babel/foreign/begin}{default}
4587 \AssignTaggingSocketPlug{babel/foreign/end}{default}
4588 \ExplSyntaxOff
4589 (/tagging)

```

7. 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`

```

4590 (*kernel)
4591 \let\bbl@onlyswitch\@empty
4592 \input babel.def
4593 \let\bbl@onlyswitch\@undefined
4594 (/kernel)

```

8. Error messages

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

```

4595 (*errors)
4596 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4597 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4598 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4599 \catcode`\@=11 \catcode`\^=7
4600 %
4601 \ifx\MessageBreak\@undefined
4602 \gdef\bbl@error@i#1#2{%
4603 \begingroup
4604 \newlinechar=`^^J
4605 \def\{^^J(babel) }%
4606 \errhelp{#2}\errmessage{\{#1}%
4607 \endgroup}
4608 \else
4609 \gdef\bbl@error@i#1#2{%
4610 \begingroup
4611 \def\{\MessageBreak}%
4612 \PackageError{babel}{#1}{#2}%
4613 \endgroup}
4614 \fi
4615 \def\bbl@errmessage#1#2#3{%
4616 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4617 \bbl@error@i{#2}{#3}}
4618 % Implicit #2#3#4:
4619 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4620 %
4621 \bbl@errmessage{not-yet-available}
4622 {Not yet available}%
4623 {Find an armchair, sit down and wait}
4624 \bbl@errmessage{bad-package-option}%

```

```

4625 {Bad option '#1=#2'. Either you have misspelled the\\%
4626 key or there is a previous setting of '#1'. Valid\\%
4627 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4628 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4629 {See the manual for further details.}
4630 \bbl@errmessage{base-on-the-fly}
4631 {For a language to be defined on the fly 'base'\\%
4632 is not enough, and the whole package must be\\%
4633 loaded. Either delete the 'base' option or\\%
4634 request the languages explicitly}%
4635 {See the manual for further details.}
4636 \bbl@errmessage{undefined-language}
4637 {You haven't defined the language '#1' yet.\\%
4638 Perhaps you misspelled it or your installation\\%
4639 is not complete}%
4640 {Your command will be ignored, type <return> to proceed}
4641 \bbl@errmessage{invalid-ini-name}
4642 {'#1' not valid with the 'ini' mechanism.\\%
4643 I think you want '#2' instead. You may continue,\\%
4644 but you should fix the name. See the babel manual\\%
4645 for the available locales with 'provide'}%
4646 {See the manual for further details.}
4647 \bbl@errmessage{shorthand-is-off}
4648 {I can't declare a shorthand turned off (\string#2)}
4649 {Sorry, but you can't use shorthands which have been\\%
4650 turned off in the package options}
4651 \bbl@errmessage{not-a-shorthand}
4652 {The character '\string #1' should be made a shorthand character;\\%
4653 add the command \string\usesshorthands\string{#1\string} to
4654 the preamble.\\%
4655 I will ignore your instruction}%
4656 {You may proceed, but expect unexpected results}
4657 \bbl@errmessage{not-a-shorthand-b}
4658 {I can't switch '\string#2' on or off--not a shorthand\\%
4659 This character is not a shorthand. Maybe you made\\%
4660 a typing mistake?}%
4661 {I will ignore your instruction.}
4662 \bbl@errmessage{unknown-attribute}
4663 {The attribute #2 is unknown for language #1.}%
4664 {Your command will be ignored, type <return> to proceed}
4665 \bbl@errmessage{missing-group}
4666 {Missing group for string \string#1}%
4667 {You must assign strings to some category, typically\\%
4668 captions or extras, but you set none}
4669 \bbl@errmessage{only-lua-xe}
4670 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4671 {Consider switching to these engines.}
4672 \bbl@errmessage{only-lua}
4673 {This macro is available only in LuaLaTeX}%
4674 {Consider switching to that engine.}
4675 \bbl@errmessage{unknown-provide-key}
4676 {Unknown key '#1' in \string\babelprovide}%
4677 {See the manual for valid keys}%
4678 \bbl@errmessage{unknown-mapfont}
4679 {Option '\bbl@KVP@mapfont' unknown for\\%
4680 mapfont. Use 'direction'}%
4681 {See the manual for details.}
4682 \bbl@errmessage{no-ini-file}
4683 {There is no ini file for the requested language\\%
4684 (#1: \language). Perhaps you misspelled it or your\\%
4685 installation is not complete}%
4686 {Fix the name or reinstall babel.}
4687 \bbl@errmessage{digits-is-reserved}

```

```

4688 {The counter name 'digits' is reserved for mapping\\%
4689 decimal digits}%
4690 {Use another name.}
4691 \bbl@errmessage{limit-two-digits}
4692 {Currently two-digit years are restricted to the\\5
4693 range 0-9999}%
4694 {There is little you can do. Sorry.}
4695 \bbl@errmessage{counter-too-large}
4696 {Counter too large (#1)}%
4697 {Currently this is the limit.}
4698 \bbl@errmessage{no-ini-info}
4699 {I've found no info for the current locale.\\%
4700 The corresponding ini file has not been loaded\\%
4701 Perhaps it doesn't exist}%
4702 {See the manual for details.}
4703 \bbl@errmessage{unknown-ini-field}
4704 {Unknown field '#1' in \string\BCPdata.\\%
4705 Perhaps you misspelled it}%
4706 {See the manual for details.}
4707 \bbl@errmessage{unknown-locale-key}
4708 {Unknown key for locale '#2':\\%
4709 #3\\%
4710 \string#1 will be set to \string\relax}%
4711 {Perhaps you misspelled it.}%
4712 \bbl@errmessage{adjust-only-vertical}
4713 {Currently, #1 related features can be adjusted only\\%
4714 in the main vertical list}%
4715 {Maybe things change in the future, but this is what it is.}
4716 \bbl@errmessage{layout-only-vertical}
4717 {Currently, layout related features can be adjusted only\\%
4718 in vertical mode}%
4719 {Maybe things change in the future, but this is what it is.}
4720 \bbl@errmessage{bidi-only-lua}
4721 {The bidi method 'basic' is available only in\\%
4722 luatex. I'll continue with 'bidi=default', so\\%
4723 expect wrong results.\\%
4724 Suggested actions:\\%
4725 * If possible, switch to luatex, as xetex is not\\%
4726 recommend anymore.\\
4727 * If you can't, try 'bidi=bidi' with xetex.\\%
4728 * With pdftex, only 'bidi=default' is available.}%
4729 {See the manual for further details.}
4730 \bbl@errmessage{multiple-bidi}
4731 {Multiple bidi settings inside a group\\%
4732 I'll insert a new group, but expect wrong results.\\%
4733 Suggested action:\\%
4734 * Add a new group where appropriate.}
4735 {See the manual for further details.}
4736 \bbl@errmessage{unknown-package-option}
4737 {Unknown option '\CurrentOption'.\\%
4738 Suggested actions:\\%
4739 * Make sure you haven't misspelled it\\%
4740 * Check in the babel manual that it's supported\\%
4741 * If supported and it's a language, you may\\%
4742 \space\space need in some distributions a separate\\%
4743 \space\space installation\\%
4744 * If installed, check there isn't an old\\%
4745 \space\space version of the required files in your system\\%
4746 * If it's an unsupported language, create it with\\%
4747 \space\space \string\babelprovide\space (see the manual)}
4748 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4749 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4750 headfoot=, strings=, config=, hyphenmap=, or a language name.}

```

```

4751 \bbl@errmessage{config-not-found}
4752 {Local config file '\bbl@opt@config.cfg' not found.\\%
4753   Suggested actions:\\%
4754   * Make sure you haven't misspelled it in config=\\%
4755   * Check it exists and it's in the correct path}%
4756 {Perhaps you misspelled it.}
4757 \bbl@errmessage{late-after-babel}
4758 {Too late for \string\AfterBabelLanguage}%
4759 {Languages have been loaded, so I can do nothing}
4760 \bbl@errmessage{double-hyphens-class}
4761 {Double hyphens aren't allowed in \string\babelcharclass\\%
4762   because it's potentially ambiguous}%
4763 {See the manual for further info}
4764 \bbl@errmessage{unknown-interchar}
4765 {'#1' for '\language' cannot be enabled.\\%
4766   Maybe there is a typo}%
4767 {See the manual for further details.}
4768 \bbl@errmessage{unknown-interchar-b}
4769 {'#1' for '\language' cannot be disabled.\\%
4770   Maybe there is a typo}%
4771 {See the manual for further details.}
4772 \bbl@errmessage{charproperty-only-vertical}
4773 {\string\babelcharproperty\space can be used only in\\%
4774   vertical mode (preamble or between paragraphs)}%
4775 {See the manual for further info}
4776 \bbl@errmessage{unknown-char-property}
4777 {No property named '#2'. Allowed values are\\%
4778   direction (bc), mirror (bmg), and linebreak (lb)}%
4779 {See the manual for further info}
4780 \bbl@errmessage{bad-transform-option}
4781 {Bad option '#1' in a transform.\\%
4782   I'll ignore it but expect more errors}%
4783 {See the manual for further info.}
4784 \bbl@errmessage{font-conflict-transforms}
4785 {Transforms cannot be re-assigned to different\\%
4786   fonts. The conflict is in '\bbl@kv@label'.\\%
4787   Apply the same fonts or use a different label}%
4788 {See the manual for further details.}
4789 \bbl@errmessage{transform-not-available}
4790 {'#1' for '\language' cannot be enabled.\\%
4791   Maybe there is a typo or it's a font-dependent transform}%
4792 {See the manual for further details.}
4793 \bbl@errmessage{transform-not-available-b}
4794 {'#1' for '\language' cannot be disabled.\\%
4795   Maybe there is a typo or it's a font-dependent transform}%
4796 {See the manual for further details.}
4797 \bbl@errmessage{year-out-range}
4798 {Year out of range.\\%
4799   The allowed range is #1}%
4800 {See the manual for further details.}
4801 \bbl@errmessage{only-pdfTeX-lang}
4802 {The '#1' ldf style doesn't work with #2,\\%
4803   but you can use the ini locale instead.\\%
4804   Try adding 'provide=*' to the option list. You may\\%
4805   also want to set 'bidi=' to some value}%
4806 {See the manual for further details.}
4807 \bbl@errmessage{hyphenmins-args}
4808 {\string\babelhyphenmins\ accepts either the optional\\%
4809   argument or the star, but not both at the same time}%
4810 {See the manual for further details.}
4811 \bbl@errmessage{no-locale-for-meta}
4812 {There isn't currently a locale for the 'lang' requested\\%
4813   in the PDF metadata ('#1'). To fix it, you can\\%

```

```

4814 set explicitly a similar language (using the same\\%
4815 script) with the key main= when loading babel. If you\\%
4816 continue, I'll fallback to the 'nil' language, with\\%
4817 tag 'und' and script 'Latn', but expect a bad font\\%
4818 rendering with other scripts. You may also need set\\%
4819 explicitly captions and date, too}%
4820 {See the manual for further details.}
4821 {/errors}
4822 {*patterns}

```

9. Loading hyphenation patterns

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

```

4823 <@Make sure ProvidesFile is defined@>
4824 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4825 \xdef\bbl@format{\jobname}
4826 \def\bbl@version{<@version@>}
4827 \def\bbl@date{<@date@>}
4828 \ifx\AtBeginDocument\@undefined
4829 \def\@empty{}
4830 \fi
4831 <@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.

```

4832 \def\process@line#1#2 #3 #4 {%
4833 \ifx=#1%
4834 \process@synonym{#2}%
4835 \else
4836 \process@language{#1#2}{#3}{#4}%
4837 \fi
4838 \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.

```

4839 \toks@{}
4840 \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.

```

4841 \def\process@synonym#1{%
4842 \ifnum\last@language=\m@ne
4843 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4844 \else
4845 \expandafter\chardef\csname l@#1\endcsname\last@language
4846 \wlog{\string\l@#1=\string\language\the\last@language}%
4847 \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
4848 \csname\language\endcsname hyphenmins\endcsname
4849 \let\bbl@elt\relax
4850 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4851 \fi}

```

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

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

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

Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. T_EX 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.

```

4852 \def\process@language#1#2#3{%
4853   \expandafter\addlanguage\csname l@#1\endcsname
4854   \expandafter\language\csname l@#1\endcsname
4855   \edef\language#1{%
4856     \bbl@hook@everylanguage{#1}%
4857     % > luatex
4858     \bbl@get@enc#1::\@@@
4859     \begingroup
4860       \lefthyphenmin\m@ne
4861       \bbl@hook@loadpatterns{#2}%
4862       % > luatex
4863       \ifnum\lefthyphenmin=\m@ne
4864         \else
4865           \expandafter\xdef\csname #1hyphenmins\endcsname{%
4866             \the\lefthyphenmin\the\righthyphenmin}%
4867         \fi
4868       \endgroup
4869     \def\bbl@tempa{#3}%
4870     \ifx\bbl@tempa\@empty\else
4871       \bbl@hook@loadexceptions{#3}%
4872       % > luatex
4873     \fi
4874     \let\bbl@elt\relax
4875     \edef\bbl@languages{%
4876       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4877     \ifnum\the\language=\z@
4878       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4879         \set@hyphenmins\tw@\thr@@\relax
4880       \else
4881         \expandafter\expandafter\expandafter\set@hyphenmins
4882         \csname #1hyphenmins\endcsname
4883       \fi
4884       \the\toks@
4885       \toks@{}%
4886     \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.

```
4887 \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.

```
4888 \def\bbl@hook@everylanguage#1{}
4889 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4890 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4891 \def\bbl@hook@loadkernel#1{%
4892   \def\addlanguage{\csname newlanguage\endcsname}%
4893   \def\adddialect##1##2{%
4894     \global\chardef##1##2\relax
4895     \wlog{\string##1 = a dialect from \string\language##2}}%
4896   \def\iflanguage##1{%
4897     \expandafter\ifx\csname l@##1\endcsname\relax
4898       \nolannerr{##1}%
4899     \else
4900       \ifnum\csname l@##1\endcsname=\language
4901         \expandafter\expandafter\expandafter\@firstoftwo
4902       \else
4903         \expandafter\expandafter\expandafter\@secondoftwo
4904       \fi
4905     \fi}%
4906   \def\providehyphenmins##1##2{%
4907     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4908       \namedef{##1hyphenmins}{##2}%
4909     \fi}%
4910   \def\set@hyphenmins##1##2{%
4911     \lefthyphenmin##1\relax
4912     \righthyphenmin##2\relax}%
4913   \def\selectlanguage{%
4914     \errhelp{Selecting a language requires a package supporting it}%
4915     \errmessage{No multilingual package has been loaded}}%
4916   \let\foreignlanguage\selectlanguage
4917   \let\otherlanguage\selectlanguage
4918   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4919   \def\bbl@usehooks##1##2{%
4920     \def\setlocale{%
4921       \errhelp{Find an armchair, sit down and wait}%
4922       \errmessage{(babel) Not yet available}}%
4923     \let\uselocale\setlocale
4924     \let\locale\setlocale
4925     \let\selectlocale\setlocale
4926     \let\localename\setlocale
4927     \let\textlocale\setlocale
4928     \let\textlanguage\setlocale
4929     \let\languagetext\setlocale}
4930   \begingroup
4931     \def\AddBabelHook#1#2{%
4932       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4933         \def\next{\toks1}%
4934       \else
4935         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4936       \fi
4937     \next}
4938   \ifx\directlua\@undefined
4939     \ifx\XeTeXinputencoding\@undefined\else
4940       \input xebabel.def
4941     \fi
4942   \else
```

```

4943 \input luababel.def
4944 \fi
4945 \openin1 = babel-\bbl@format.cfg
4946 \ifeof1
4947 \else
4948 \input babel-\bbl@format.cfg\relax
4949 \fi
4950 \closein1
4951 \endgroup
4952 \bbl@hook@loadkernel{switch.def}

```

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

```

4953 \openin1 = language.dat

```

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

```

4954 \def\language#1{english}%
4955 \ifeof1
4956 \message{I couldn't find the file language.dat,\space
4957          I will try the file hyphen.tex}
4958 \input hyphen.tex\relax
4959 \chardef\l@english\z@
4960 \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`.

```

4961 \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.

```

4962 \loop
4963 \endlinechar\m@ne
4964 \read1 to \bbl@line
4965 \endlinechar\^M

```

If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of `\bbl@line`. This is needed to be able to recognize the arguments of `\process@line` later on. The default language should be the very first one.

```

4966 \if T\ifeof1\fi T\relax
4967 \ifx\bbl@line\empty\else
4968 \edef\bbl@line{\bbl@line\space\space\space}%
4969 \expandafter\process@line\bbl@line\relax
4970 \fi
4971 \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.

```

4972 \begingroup
4973 \def\bbl@elt#1#2#3#4{%
4974 \global\language=#2\relax
4975 \gdef\language#1}%
4976 \def\bbl@elt##1##2##3##4{}}%
4977 \bbl@languages
4978 \endgroup
4979 \fi
4980 \closein1

```

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

```

4981 \if/\the\toks@/\else

```

```

4982 \errhelp{language.dat loads no language, only synonyms}
4983 \errmessage{Orphan language synonym}
4984 \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.

```

4985 \let\bbl@line\@undefined
4986 \let\process@line\@undefined
4987 \let\process@synonym\@undefined
4988 \let\process@language\@undefined
4989 \let\bbl@get@enc\@undefined
4990 \let\bbl@hyph@enc\@undefined
4991 \let\bbl@tempa\@undefined
4992 \let\bbl@hook@loadkernel\@undefined
4993 \let\bbl@hook@everylanguage\@undefined
4994 \let\bbl@hook@loadpatterns\@undefined
4995 \let\bbl@hook@loadexceptions\@undefined
4996 (/patterns)

```

Here the code for `initTeX` ends.

10. luatex + xetex: common stuff

Add the bidi handler just before `luaotfload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi (although default also applies to `pdftex`).

```

4997 (<*More package options>) ≡
4998 \chardef\bbl@bidimode\z@
4999 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
5000 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
5001 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
5002 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
5003 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
5004 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
5005 \DeclareOption{bidi=unibidi}{\chardef\bbl@bidimode=301 }
5006 (/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`.

```

5007 (<*Font selection>) ≡
5008 \bbl@trace{Font handling with fontspec}
5009 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5010 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5011 \DisableBabelHook{babel-fontspec}
5012 \@onlypreamble\babelfont
5013 \ifx\NewDocumentCommand\@undefined\else % Not plain
5014   \NewDocumentCommand\babelfont{0}{m0}{m0}}{%
5015     \bbl@bblfont@o{#1}{#2}{#3,#5}{#4}}
5016 \fi
5017 \newcommand\bbl@bblfont@o[2][]{% 1=langs/scripts 2=fam
5018   \ifx\fontspec\@undefined
5019     \usepackage{fontspec}%
5020   \fi
5021   \EnableBabelHook{babel-fontspec}%
5022   \edef\bbl@tempa{#1}%
5023   \def\bbl@tempb{#2}% Used by \bbl@bblfont
5024   \bbl@bblfont}
5025 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
5026   \bbl@ifunset{\bbl@tempb family}%
5027   {\bbl@providfam{\bbl@tempb}}%
5028   }%

```

```

5029 % For the default font, just in case:
5030 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
5031 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
5032 {\bbl@csarg\edef{\bbl@tempb dflt@{<#1>{#2}}}% save bbl@rmdflt@
5033 \bbl@exp{%
5034 \let<\bbl@tempb dflt@\language>\<\bbl@tempb dflt@>%
5035 \\\bbl@font@set<\bbl@tempb dflt@\language>%
5036 \<\bbl@tempb default>\<\bbl@tempb family>}}%
5037 {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
5038 \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:

```

5039 \def\bbl@providefam#1{%
5040 \bbl@exp{%
5041 \\\newcommand<#1default>{}% Just define it
5042 \\\bbl@add@list\\bbl@font@fams{#1}%
5043 \\\NewHook{#1family}%
5044 \\\DeclareRobustCommand<#1family>{%
5045 \\\not@math@alphabet<#1family>\relax
5046 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
5047 \\\fontfamily<#1default>%
5048 \\\UseHook{#1family}%
5049 \\\selectfont}%
5050 \\\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.

```

5051 \def\bbl@nostdfont#1{%
5052 \bbl@once{nostdfam-\f@family}%
5053 {\bbl@infowarn{The current font is not a babel standard family:\\%
5054 #1%
5055 \fontname\font\\%
5056 There is nothing intrinsically wrong, and you can\\%,
5057 ignore this message altogether if you do not need\\%
5058 this font. If they are used in the document, be aware\\%
5059 'babel' will not set Script and Language for it, so\\%
5060 you may consider defining a new family with \string\babelfont.\\%
5061 See the manual for further details about \string\babelfont.
5062 Reported}}%
5063 {}}%
5064 \gdef\bbl@switchfont{%
5065 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
5066 \bbl@exp{% e.g., Arabic -> arabic
5067 \lowercase{\edef\\bbl@tempa{\bbl@ccl{sname}}}}%
5068 \bbl@foreach\bbl@font@fams{%
5069 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
5070 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
5071 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
5072 {}% 123=F - nothing!
5073 {\bbl@exp{% 3=T - from generic
5074 \global\let<\bbl@##1dflt@\language>%
5075 \<\bbl@##1dflt@>}}}%
5076 {\bbl@exp{% 2=T - from script
5077 \global\let<\bbl@##1dflt@\language>%
5078 \<\bbl@##1dflt@*\bbl@tempa>}}}%
5079 {}}% 1=T - language, already defined
5080 \def\bbl@tempa{\bbl@nostdfont}}%
5081 \bbl@foreach\bbl@font@fams{% don't gather with prev for
5082 \bbl@ifunset{\bbl@##1dflt@\language}%
5083 {\bbl@cs{famrst@##1}%
5084 \global\bbl@csarg\let{famrst@##1}\relax}%
5085 {\bbl@exp{% order is relevant.
5086 \\\bbl@add\\originalTeX{%
5087 \\\bbl@font@rst{\bbl@ccl{##1dflt}}}%

```

```

5088             \<##ldefault>\<##lfamily>{##l}}%
5089             \\bbl@font@set\<bbl@##ldflt@\language>% the main part!
5090             \<##ldefault>\<##lfamily>}}}%
5091 \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`.

```

5092 \ifx\f@family\undefined\else % if latex
5093 \ifcase\bbl@engine % if pdftex
5094 \let\bbl@cckstdfont\relax
5095 \else
5096 \def\bbl@cckstdfont{%
5097 \begingroup
5098 \global\let\bbl@cckstdfont\relax
5099 \let\bbl@tempa\empty
5100 \bbl@foreach\bbl@font@fams{%
5101 \bbl@ifunset\bbl@##ldflt@}%
5102 {\@nameuse{##lfamily}%
5103 \bbl@csarg\gdef{WFF@f@family}}}% Flag
5104 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##lfamily>= f@family\\}%
5105 \space\space\fontname\font\\}%
5106 \bbl@csarg\xdef{##ldflt@}{f@family}%
5107 \expandafter\xdef\csname ##ldefault\endcsname{f@family}}}%
5108 {}}%
5109 \ifx\bbl@tempa\empty\else
5110 \bbl@infowarn{The following font families will use the default\\%
5111 settings for all or some languages:\\%
5112 \bbl@tempa
5113 There is nothing intrinsically wrong with it, but\\%
5114 'babel' will no set Script and Language, which could\\%
5115 be relevant in some languages. If your document uses\\%
5116 these families, consider redefining them with \string\babelfont.\\%
5117 Reported}%
5118 \fi
5119 \endgroup}
5120 \fi
5121 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

5122 \def\bbl@font@set#1#2#3{ e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
5123 \bbl@xin@{<>}{#1}%
5124 \ifin@
5125 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
5126 \fi
5127 \bbl@exp{%
5128 \def\\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
5129 \\bbl@ifsamestring{#2}{f@family}%
5130 {\\#3%
5131 \\bbl@ifsamestring{f@series}{\bfdefault}{\\bfseries}}}%
5132 \let\\bbl@tempa\relax}%
5133 {}}}

```

Loaded locally, which does its job, but very must be global. The problem is how. This actually defines a font predeclared with `\babelfont`, making sure `Script` and `Language` names are defined. If they are not, the corresponding data in the ini file is used. The font is actually set temporarily to get

the family name (\f@family). There is also a hack because by default some replacements related to the bold series are sometimes assigned to the wrong font (see issue #92).

```

5134 \def\bbl@fontspec@lsys{%
5135   \ifx\newfontscript\undefined\else\bbl@exp{%
5136     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
5137     {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
5138     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
5139     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}}%
5140   \fi}
5141 \ifx\AddToHook\undefined
5142   \let\bbl@fontspec@lsys\relax
5143 \else
5144   \AddToHook{selectfont}{\bbl@fontspec@lsys}
5145 \fi
5146 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
5147   \let\bbl@tempe\bbl@mapselect
5148   \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
5149   \bbl@exp{\bbl@replace{\bbl@tempb{\bbl@stripslash\family/}}}%
5150   \let\bbl@mapselect\relax
5151   \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
5152   \let#4\empty % Make sure \renewfontfamily is valid
5153   \bbl@set@renderer
5154   \bbl@fontspec@lsys
5155   \bbl@exp{%
5156     \let\bbl@temp@pfam\<\bbl@stripslash#4\space>% e.g., '\rmfamily '
5157     \renewfontfamily\#4%
5158     [\bbl@cl{lsys},% xetex removes unknown features :-(
5159     \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
5160     #2]}{#3}% i.e., \bbl@exp{..}{#3}
5161   \bbl@unset@renderer
5162   \begingroup
5163     #4%
5164     \xdef#1{\f@family}% e.g., \bbl@rmdflt@lang{FreeSerif(0)}
5165   \endgroup
5166   \bbl@xin@{\detokenize{>ssub*}}%
5167   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
5168   \ifin@
5169     \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
5170   \fi
5171   \bbl@xin@{\detokenize{>ssub*}}%
5172   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
5173   \ifin@
5174     \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
5175   \fi
5176   \let#4\bbl@temp@fam
5177   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
5178   \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.

```

5179 \def\bbl@font@rst#1#2#3#4{%
5180   \bbl@ccarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

5181 \def\bbl@font@fams{rm,sf,tt}
5182 {/Font selection}

```

11. Hooks for XeTeX and LuaTeX

11.1. XeTeX

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

Now, the code.

```
5183 (*xetex)
5184 \def\BabelStringsDefault{unicode}
5185 \let\xebbl@stop\relax
5186 \AddBabelHook{xetex}{encodedcommands}{%
5187   \def\bbl@tempa{#1}%
5188   \ifx\bbl@tempa\@empty
5189     \XeTeXinputencoding"bytes"%
5190   \else
5191     \XeTeXinputencoding"#1"%
5192   \fi
5193   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
5194 \AddBabelHook{xetex}{stopcommands}{%
5195   \xebbl@stop
5196   \let\xebbl@stop\relax}
5197 \def\bbl@input@classes{% Used in CJK intraspaces
5198   \input{load-unicode-xetex-classes.tex}%
5199   \let\bbl@input@classes\relax}
5200 \def\bbl@intraspace#1 #2 #3\@{%
5201   \bbl@csarg\gdef{xeisp@\languagename}%
5202     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
5203 \def\bbl@intrapenalty#1\@{%
5204   \bbl@csarg\gdef{xeipn@\languagename}%
5205     {\XeTeXlinebreakpenalty #1\relax}}
5206 \def\bbl@provide@intraspace{%
5207   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
5208   \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{lnbrk}} \fi
5209   \ifin@
5210     \bbl@ifunset{bbl@intsp@\languagename}{%
5211       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5212         \ifx\bbl@KVP@intraspace\@nnil
5213           \bbl@exp{%
5214             \\bbl@intraspace\bbl@cl{intsp}\\\@}%
5215           \fi
5216           \ifx\bbl@KVP@intrapenalty\@nnil
5217             \bbl@intrapenalty0\@
5218           \fi
5219         \fi
5220         \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
5221           \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
5222         \fi
5223         \ifx\bbl@KVP@intrapenalty\@nnil\else
5224           \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5225         \fi
5226         \bbl@exp{%
5227           \\bbl@add<extras\languagename>{%
5228             \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
5229             \<bbl@xeisp@\languagename>%
5230             \<bbl@xeipn@\languagename>%
5231             \\bbl@tglobal\<extras\languagename>%
5232             \\bbl@add<noextras\languagename>{%
5233               \XeTeXlinebreaklocale ""}%
5234             \\bbl@tglobal\<noextras\languagename>%
5235             \ifx\bbl@ispacesize\@undefined
5236               \gdef\bbl@ispacesize{\bbl@cl{xeisp}}}%
5237             \ifx\AtBeginDocument\@notprerr
5238               \expandafter\@secondoftwo % to execute right now
```

```

5239      \fi
5240      \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
5241      \fi}%
5242  \fi}
5243  \ifx\DisableBabelHook\undefined\endinput\fi
5244  \let\bbl@set@renderer\relax
5245  \let\bbl@unset@renderer\relax
5246  <@Font selection>
5247  \def\bbl@provide@extra#1{}

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

5248  \def\bbl@xenoxyph@d{%
5249    \bbl@ifset{\bbl@prehc@language}%
5250    {\ifnum\hyphenchar\font=\defaultthyphenchar
5251      \iffontchar\font\bbl@cl{prehc}\relax
5252      \hyphenchar\font\bbl@cl{prehc}\relax
5253      \else\iffontchar\font"200B
5254        \hyphenchar\font"200B
5255      \else
5256        \bbl@warning
5257          {Neither 0 nor ZERO WIDTH SPACE are available\\%
5258            in the current font, and therefore the hyphen\\%
5259            will be printed. Try changing the fontspec's\\%
5260            'HyphenChar' to another value, but be aware\\%
5261            this setting is not safe (see the manual).\\%
5262            Reported}%
5263        \hyphenchar\font\defaultthyphenchar
5264      \fi\fi
5265    \fi}%
5266    {\hyphenchar\font\defaultthyphenchar}}

```

11.2. 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.

```

5267  \ifnum\xe@alloc@intercharclass<\thr@@
5268    \xe@alloc@intercharclass\thr@@
5269  \fi
5270  \chardef\bbl@xe@class@default=\z@
5271  \chardef\bbl@xe@class@cjkideogram=\@ne
5272  \chardef\bbl@xe@class@cjkleftpunctuation=\tw@
5273  \chardef\bbl@xe@class@cjkrightpunctuation=\thr@@
5274  \chardef\bbl@xe@class@boundary=4095
5275  \chardef\bbl@xe@class@ignore=4096

```

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

```

5276  \AddBabelHook{babel-interchar}{beforeextras}{%
5277    \nameuse{\bbl@xe@chars@language}}
5278  \DisableBabelHook{babel-interchar}
5279  \protected\def\bbl@charclass#1{%
5280    \ifnum\count@<\z@
5281      \count@=-\count@
5282    \loop
5283      \bbl@exp{%
5284        \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5285        \XeTeXcharclass\count@ \bbl@tempc
5286        \ifnum\count@<`#1\relax
5287          \advance\count@\@ne
5288        \repeat
5289    \else

```

```

5290 \babel@savevariable{\XeTeXcharclass`#1}%
5291 \XeTeXcharclass`#1 \bbl@tempc
5292 \fi
5293 \count@`#1\relax}

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

5294 \newcommand\bbl@ifinterchar[1]{%
5295 \let\bbl@tempa\@gobble % Assume to ignore
5296 \edef\bbl@tempb{\zap@space#1 \@empty}%
5297 \ifx\bbl@KVP@interchar\@nnil\else
5298 \bbl@replace\bbl@KVP@interchar{ }{,}%
5299 \bbl@foreach\bbl@tempb{%
5300 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5301 \ifin@
5302 \let\bbl@tempa\@firstofone
5303 \fi}%
5304 \fi
5305 \bbl@tempa}
5306 \newcommand\IfBabelIntercharT[2]{%
5307 \bbl@carg\bbl@add{\bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5308 \newcommand\babelcharclass[3]{%
5309 \EnableBabelHook{babel-interchar}%
5310 \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5311 \def\bbl@tempb##1{%
5312 \ifx##1\@empty\else
5313 \ifx##1-%
5314 \bbl@upto
5315 \else
5316 \bbl@charclass{%
5317 \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5318 \fi
5319 \expandafter\bbl@tempb
5320 \fi}%
5321 \bbl@ifunset{\bbl@xechars@#1}%
5322 {\toks@{%
5323 \babel@savevariable\XeTeXinterchartokenstate
5324 \XeTeXinterchartokenstate\@ne
5325 }}%
5326 {\toks@\expandafter\expandafter\expandafter{%
5327 \csname bbl@xechars@#1\endcsname}}%
5328 \bbl@csarg\edef{xechars@#1}{%
5329 \the\toks@
5330 \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5331 \bbl@tempb#3\@empty}}
5332 \protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
5333 \protected\def\bbl@upto{%
5334 \ifnum\count@>\z@
5335 \advance\count@\@ne
5336 \count@-\count@
5337 \else\ifnum\count@=\z@
5338 \bbl@charclass{-}%
5339 \else
5340 \bbl@error{double-hyphens-class}{ }{}{}%
5341 \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>.

```

5342 \def\bbl@ignoreinterchar{%

```

```

5343 \ifnum\language=\l@nohyphenation
5344 \expandafter\@gobble
5345 \else
5346 \expandafter\@firstofone
5347 \fi}
5348 \newcommand\babelinterchar[5][]{%
5349 \let\bbl@kv@label\@empty
5350 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}}%
5351 \namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5352 {\bbl@ignoreinterchar{#5}}}%
5353 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5354 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5355 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5356 \XeTeXinterchartoks
5357 \@nameuse{\bbl@xeclasse@\bbl@tempa @#2}
5358 \bbl@ifunset{\bbl@xeclasse@\bbl@tempa @#2}{#2} %
5359 \@nameuse{\bbl@xeclasse@\bbl@tempb @#2}
5360 \bbl@ifunset{\bbl@xeclasse@\bbl@tempb @#2}{#2} %
5361 = \expandafter%
5362 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5363 \csname\zap@space bbl@xeinter@\bbl@kv@label
5364 @#3@#4@#2 \@empty\endcsname}}}}
5365 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5366 \bbl@ifunset{\bbl@ic@#1\@languagename}%
5367 {\bbl@error{unknown-interchar}{#1}{}}}%
5368 {\bbl@csarg\let{ic@#1\@languagename}\@firstofone}}
5369 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5370 \bbl@ifunset{\bbl@ic@#1\@languagename}%
5371 {\bbl@error{unknown-interchar-b}{#1}{}}}%
5372 {\bbl@csarg\let{ic@#1\@languagename}\@gobble}}
5373 {/xetex}

```

11.3. 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 T_EX 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 pdf_{TE}X and xetex.

```

5374 {*xetex|texxet}
5375 \providecommand\bbl@provide@intraspace{}
5376 \bbl@trace{Redefinitions for bidi layout}

Finish here if there in no layout.

5377 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5378 \ifx\bbl@opt@layout\@undefined\else % Plain
5379 \IfBabelLayout{nopars}
5380 {}
5381 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5382 \fi
5383 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5384 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5385 \ifnum\bbl@bidimode>\z@
5386 \IfBabelLayout{pars}
5387 {\def\@hangfrom#1{%
5388 \setbox\@tempboxa\hbox{#1}}%
5389 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5390 \noindent\box\@tempboxa}
5391 \def\raggedright{%
5392 \let\@centercr
5393 \bbl@startskip\z@skip

```

```

5394 \@rightskip\@flushglue
5395 \bbl@endskip\@rightskip
5396 \parindent\z@
5397 \parfillskip\bbl@startskip}
5398 \def\raggedleft{%
5399 \let\\\@centercr
5400 \bbl@startskip\@flushglue
5401 \bbl@endskip\z@skip
5402 \parindent\z@
5403 \parfillskip\bbl@endskip}}
5404 {}
5405 \fi
5406 \IfBabelLayout{lists}
5407 {\bbl@sreplace\list
5408 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5409 \def\bbl@listleftmargin{%
5410 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5411 \ifcase\bbl@engine
5412 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5413 \def\p@enumiii{\p@enumii}\theenumii}%
5414 \fi
5415 \bbl@sreplace\@verbatim
5416 {\leftskip\@totalleftmargin}%
5417 {\bbl@startskip\textwidth
5418 \advance\bbl@startskip-\linewidth}%
5419 \bbl@sreplace\@verbatim
5420 {\rightskip\z@skip}%
5421 {\bbl@endskip\z@skip}}%
5422 {}
5423 \IfBabelLayout{contents}
5424 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5425 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5426 {}
5427 \IfBabelLayout{columns}
5428 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5429 \def\bbl@outputbox#1{%
5430 \hb@xt@\textwidth{%
5431 \hskip\columnwidth
5432 \hfil
5433 {\normalcolor\vrule \@width\columnseprule}%
5434 \hfil
5435 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5436 \hskip-\textwidth
5437 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5438 \hskip\columnsep
5439 \hskip\columnwidth}}}%
5440 {}

```

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.

```

5441 \IfBabelLayout{counters*}%
5442 {\bbl@add\bbl@opt@layout{.counters.}%
5443 \AddToHook{shipout/before}{%
5444 \let\bbl@tempa\babelsublr
5445 \let\babelsublr\@firstofone
5446 \let\bbl@save@thepage\thepage
5447 \protected@edef\thepage{\thepage}%
5448 \let\babelsublr\bbl@tempa}%
5449 \AddToHook{shipout/after}{%
5450 \let\thepage\bbl@save@thepage}}{}
5451 \IfBabelLayout{counters}%
5452 {\let\bbl@latinarabic=\@arabic
5453 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%

```

```

5454 \let\bbl@asciroman=\@roman
5455 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5456 \let\bbl@asciiRoman=\@Roman
5457 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5458 \fi % end if layout
5459 (/xetex|texet)

```

11.4. 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).

```

5460 (*texet)
5461 \def\bbl@provide@extra#1{%
5462 % == auto-select encoding ==
5463 \ifx\bbl@encoding@select@off\@empty\else
5464 \bbl@ifunset{\bbl@encoding@#1}%
5465 {\def\@elt##1{,##1,}%
5466 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5467 \count@z@
5468 \bbl@foreach\bbl@tempe{%
5469 \def\bbl@tempd{##1}% Save last declared
5470 \advance\count@z@one}%
5471 \ifnum\count@z@>\@ne % (1)
5472 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5473 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5474 \bbl@replace\bbl@tempa{ },}%
5475 \global\bbl@csarg\let{encoding@#1}\@empty
5476 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5477 \ifin@else % if main encoding included in ini, do nothing
5478 \let\bbl@tempb\relax
5479 \bbl@foreach\bbl@tempa{%
5480 \ifx\bbl@tempb\relax
5481 \bbl@xin@{,##1,}{,\bbl@tempe,}%
5482 \ifin@\def\bbl@tempb{##1}\fi
5483 \fi}%
5484 \ifx\bbl@tempb\relax\else
5485 \bbl@exp{%
5486 \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5487 \gdef\<bbl@encoding@#1>{%
5488 \\\babel@save\\f@encoding
5489 \\\bbl@add\\originalTeX{\\selectfont}%
5490 \\\fontencoding{\bbl@tempb}%
5491 \\\selectfont}}%
5492 \fi
5493 \fi
5494 \fi}%
5495 }%
5496 \fi}
5497 (/texet)

```

11.5. LuaTeX

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

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

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following

rule applies: if the “0th” language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, they are added, but note if the language patterns have not been preloaded they won’t at run time.

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

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

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

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

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

```

5498 (*luatex)
5499 \directlua{ Babel = Babel or {} } % DL2
5500 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5501 \bbl@trace{Read language.dat}
5502 \ifx\bbl@readstream\undefined
5503 \csname newread\endcsname\bbl@readstream
5504 \fi
5505 \begingroup
5506 \toks@{}
5507 \count@ \z@ % 0=start, 1=0th, 2=normal
5508 \def\bbl@process@line#1#2 #3 #4 {%
5509 \ifx=#1%
5510 \bbl@process@synonym{#2}%
5511 \else
5512 \bbl@process@language{#1#2}{#3}{#4}%
5513 \fi
5514 \ignorespaces}
5515 \def\bbl@manylang{%
5516 \ifnum\bbl@last>\@ne
5517 \bbl@info{Non-standard hyphenation setup}%
5518 \fi
5519 \let\bbl@manylang\relax}
5520 \def\bbl@process@language#1#2#3{%
5521 \ifcase\count@
5522 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5523 \or
5524 \count@\tw@
5525 \fi
5526 \ifnum\count@=\tw@
5527 \expandafter\addlanguage\csname l@#1\endcsname
5528 \language\allocationnumber
5529 \chardef\bbl@last\allocationnumber
5530 \bbl@manylang
5531 \let\bbl@elt\relax
5532 \xdef\bbl@languages{%
5533 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5534 \fi
5535 \the\toks@
5536 \toks@{}
5537 \def\bbl@process@synonym@aux#1#2{%
5538 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5539 \let\bbl@elt\relax

```

```

5540 \xdef\bbbl@languages{%
5541 \bbbl@languages\bbbl@elt{#1}{#2}{}}}%
5542 \def\bbbl@process@synonym#1{%
5543 \ifcase\count@
5544 \toks@\expandafter{\the\toks@\relax\bbbl@process@synonym{#1}}%
5545 \or
5546 \ifundefined{zth@#1}{\bbbl@process@synonym@aux{#1}{0}}}%
5547 \else
5548 \bbbl@process@synonym@aux{#1}{\the\bbbl@last}%
5549 \fi}
5550 \ifx\bbbl@languages\undefined % Just a (sensible?) guess
5551 \chardef\l@english\z@
5552 \chardef\l@USenglish\z@
5553 \chardef\bbbl@last\z@
5554 \global\@namedef{bbbl@hyphendata@0}{{hyphen.tex}}
5555 \gdef\bbbl@languages{%
5556 \bbbl@elt{english}{0}{hyphen.tex}}%
5557 \bbbl@elt{USenglish}{0}{}}
5558 \else
5559 \global\let\bbbl@languages@format\bbbl@languages
5560 \def\bbbl@elt#1#2#3#4{% Remove all except language 0
5561 \ifnum#2>\z@\else
5562 \noexpand\bbbl@elt{#1}{#2}{#3}{#4}%
5563 \fi}%
5564 \xdef\bbbl@languages{\bbbl@languages}%
5565 \fi
5566 \def\bbbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
5567 \bbbl@languages
5568 \openin\bbbl@readstream=language.dat
5569 \ifeof\bbbl@readstream
5570 \bbbl@warning{I couldn't find language.dat. No additional\\%
5571 patterns loaded. Reported}%
5572 \else
5573 \loop
5574 \endlinechar\m@ne
5575 \read\bbbl@readstream to \bbbl@line
5576 \endlinechar``^^M
5577 \if T\ifeof\bbbl@readstream F\fi T\relax
5578 \ifx\bbbl@line\empty\else
5579 \edef\bbbl@line{\bbbl@line\space\space\space}%
5580 \expandafter\bbbl@process@line\bbbl@line\relax
5581 \fi
5582 \repeat
5583 \fi
5584 \closein\bbbl@readstream
5585 \endgroup
5586 \bbbl@trace{Macros for reading patterns files}
5587 \def\bbbl@get@enc#1:#2:#3\@@{\def\bbbl@hyph@enc{#2}}
5588 \ifx\babelcatcodetablenum\undefined
5589 \ifx\newcatcodetable\undefined
5590 \def\babelcatcodetablenum{5211}
5591 \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5592 \else
5593 \newcatcodetable\babelcatcodetablenum
5594 \newcatcodetable\bbbl@pattcodes
5595 \fi
5596 \else
5597 \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5598 \fi
5599 \def\bbbl@luapatterns#1#2{%
5600 \bbbl@get@enc#1:.\@@@
5601 \setbox\z@\hbox\bgroup
5602 \beginingroup

```

```

5603 \savecatcodetable\babelcatcodetablenum\relax
5604 \initcatcodetable\bbl@pattcodes\relax
5605 \catcodetable\bbl@pattcodes\relax
5606 \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5607 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5608 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5609 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5610 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5611 \catcode\`=12 \catcode\'=12 \catcode\"=12
5612 \input #1\relax
5613 \catcodetable\babelcatcodetablenum\relax
5614 \endgroup
5615 \def\bbl@tempa{#2}%
5616 \ifx\bbl@tempa\empty\else
5617 \input #2\relax
5618 \fi
5619 \egroup}%
5620 \def\bbl@patterns@lua#1{%
5621 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5622 \csname l@#1\endcsname
5623 \edef\bbl@tempa{#1}%
5624 \else
5625 \csname l@#1:\f@encoding\endcsname
5626 \edef\bbl@tempa{#1:\f@encoding}%
5627 \fi\relax
5628 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5629 \@ifundefined{bbl@hyphendata@the\language}%
5630 {\def\bbl@elt##1##2##3##4{%
5631 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5632 \def\bbl@tempb{##3}%
5633 \ifx\bbl@tempb\empty\else % if not a synonymous
5634 \def\bbl@tempc{##3}{##4}%
5635 \fi
5636 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5637 \fi}%
5638 \bbl@languages
5639 \@ifundefined{bbl@hyphendata@the\language}%
5640 {\bbl@info{No hyphenation patterns were set for\%
5641 language '\bbl@tempa'. Reported}}%
5642 {\expandafter\expandafter\expandafter\bbl@luapatterns
5643 \csname bbl@hyphendata@the\language\endcsname}}}%
5644 \endinput\fi

```

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

```

5645 \ifx\DisableBabelHook\@undefined
5646 \AddBabelHook{luatex}{everylanguage}{%
5647 \def\process@language##1##2##3{%
5648 \def\process@line####1####2 ####3 ####4 {}}%
5649 \AddBabelHook{luatex}{loadpatterns}{%
5650 \input #1\relax
5651 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5652 {{#1}}}%
5653 \AddBabelHook{luatex}{loadexceptions}{%
5654 \input #1\relax
5655 \def\bbl@tempb##1##2{{##1}{##1}}%
5656 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5657 {\expandafter\expandafter\expandafter\bbl@tempb
5658 \csname bbl@hyphendata@the\language\endcsname}}%
5659 \endinput\fi

```

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

```

5660 \begingroup
5661 \catcode\%=12

```

```

5662 \catcode\`=12
5663 \catcode\"=12
5664 \catcode\:=12
5665 \directlua{
5666   Babel.locale_props = Babel.locale_props or {}
5667
5668   function Babel.lua_error(e, a)
5669     tex.print([[noexpand\csname bbl@error\endcsname{]}] ..
5670       e .. '{' .. (a or '') .. '}{}{')
5671   end
5672
5673   function Babel.bytes(line)
5674     return line:gsub(".",
5675       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5676   end
5677
5678   function Babel.priority_in_callback(name,description)
5679     for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5680       if v == description then return i end
5681     end
5682     return false
5683   end
5684
5685   function Babel.begin_process_input()
5686     if luatexbase and luatexbase.add_to_callback then
5687       luatexbase.add_to_callback('process_input_buffer',
5688         Babel.bytes, 'Babel.bytes')
5689     else
5690       Babel.callback = callback.find('process_input_buffer')
5691       callback.register('process_input_buffer', Babel.bytes)
5692     end
5693   end
5694   function Babel.end_process_input ()
5695     if luatexbase and luatexbase.remove_from_callback then
5696       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5697     else
5698       callback.register('process_input_buffer', Babel.callback)
5699     end
5700   end
5701
5702   function Babel.str_to_nodes(fn, matches, base)
5703     local n, head, last
5704     if fn == nil then return nil end
5705     for s in string.utfvalues(fn(matches)) do
5706       if base.id == 7 then
5707         base = base.replace
5708       end
5709       n = node.copy(base)
5710       n.char = s
5711       if not head then
5712         head = n
5713       else
5714         last.next = n
5715       end
5716       last = n
5717     end
5718     return head
5719   end
5720
5721   Babel.linebreaking = Babel.linebreaking or {}
5722   Babel.linebreaking.before = {}
5723   Babel.linebreaking.after = {}
5724   Babel.locale = {}

```

```

5725 function Babel.linebreaking.add_before(func, pos)
5726     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5727     if pos == nil then
5728         table.insert(Babel.linebreaking.before, func)
5729     else
5730         table.insert(Babel.linebreaking.before, pos, func)
5731     end
5732 end
5733 function Babel.linebreaking.add_after(func)
5734     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5735     table.insert(Babel.linebreaking.after, func)
5736 end
5737
5738 function Babel.addpatterns(pp, lg)
5739     local lg = lang.new(lg)
5740     local pats = lang.patterns(lg) or ''
5741     lang.clear_patterns(lg)
5742     for p in pp:gmatch('[^%s]+') do
5743         ss = ''
5744         for i in string.utfcharacters(p:gsub('%d', '')) do
5745             ss = ss .. '%d?' .. i
5746         end
5747         ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5748         ss = ss:gsub('%.%%d%?$', '%%.')
5749         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5750         if n == 0 then
5751             tex.sprint(
5752                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5753                 .. p .. [[{}]])
5754             pats = pats .. ' ' .. p
5755         else
5756             tex.sprint(
5757                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5758                 .. p .. [[{}]])
5759         end
5760     end
5761     lang.patterns(lg, pats)
5762 end
5763
5764 Babel.characters = Babel.characters or {}
5765 Babel.ranges = Babel.ranges or {}
5766 function Babel.hlist_has_bidi(head)
5767     local has_bidi = false
5768     local ranges = Babel.ranges
5769     for item in node.traverse(head) do
5770         if item.id == node.id'glyph' then
5771             local itemchar = item.char
5772             local chardata = Babel.characters[itemchar]
5773             local dir = chardata and chardata.d or nil
5774             if not dir then
5775                 for nn, et in ipairs(ranges) do
5776                     if itemchar < et[1] then
5777                         break
5778                     elseif itemchar <= et[2] then
5779                         dir = et[3]
5780                         break
5781                     end
5782                 end
5783             end
5784             if dir and (dir == 'al' or dir == 'r') then
5785                 has_bidi = true
5786             end
5787         end
5788     end

```

```

5788     end
5789     return has_bidi
5790 end
5791 function Babel.set_chrnges_b (script, chrng)
5792     if chrng == '' then return end
5793     texio.write('Package babel Info: Replacing ' .. script .. ' script ranges')
5794     Babel.script_blocks[script] = {}
5795     for s, e in string.gmatch(chrng..' ', '(.-%.-%.-%s') do
5796         table.insert(
5797             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5798     end
5799 end
5800
5801 function Babel.discard_sublr(str)
5802     if str:find( [[\string\indexentry]] ) and
5803        str:find( [[\string\babelsublr]] ) then
5804         str = str:gsub( [[\string\babelsublr%s*(%b{})]]),
5805                        function(m) return m:sub(2,-2) end )
5806     end
5807     return str
5808 end
5809 }
5810 \endgroup
5811 \ifx\newattribute\undefined\else % Test for plain
5812 \newattribute\bbbl@attr@locale % DL4
5813 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbbl@attr@locale' }
5814 \AddBabelHook{luatex}{beforeextras}{%
5815     \setattribute\bbbl@attr@locale\localeid}
5816 \fi
5817 %
5818 \def\BabelStringsDefault{unicode}
5819 \let\luabbl@stop\relax
5820 \AddBabelHook{luatex}{encodedcommands}{%
5821     \def\bbbl@tempa{utf8}\def\bbbl@tempb{#1}%
5822     \ifx\bbbl@tempa\bbbl@tempb\else
5823         \directlua{Babel.begin_process_input()}%
5824         \def\luabbl@stop{%
5825             \directlua{Babel.end_process_input()}}%
5826     \fi}%
5827 \AddBabelHook{luatex}{stopcommands}{%
5828     \luabbl@stop
5829     \let\luabbl@stop\relax}
5830 %
5831 \AddBabelHook{luatex}{patterns}{%
5832     \@ifundefined{bbbl@hyphendata@the\language}%
5833     {\def\bbbl@elt##1##2##3##4{%
5834         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5835         \def\bbbl@tempb{##3}%
5836         \ifx\bbbl@tempb\empty\else % if not a synonymous
5837             \def\bbbl@tempc{##3}{##4}%
5838             \fi
5839             \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
5840         \fi}%
5841     \bbbl@languages
5842     \@ifundefined{bbbl@hyphendata@the\language}%
5843     {\bbbl@info{No hyphenation patterns were set for\%
5844         language '#2'. Reported}}%
5845     {\expandafter\expandafter\expandafter\bbbl@luapatterns
5846         \csname bbbl@hyphendata@the\language\endcsname}}}%
5847 \@ifundefined{bbbl@patterns@}{}%
5848 \beginingroup
5849     \bbbl@xin@{, \number\language, }{, \bbbl@pttnlist}%
5850 \ifin@ \else

```

```

5851 \ifx\bbl@patterns@\empty\else
5852 \directlua{ Babel.addpatterns(
5853 [[\bbl@patterns@]], \number\language) }%
5854 \fi
5855 \@ifundefined{bbl@patterns@#1}%
5856 \empty
5857 {\directlua{ Babel.addpatterns(
5858 [[\space\csname bbl@patterns@#1\endcsname]],
5859 \number\language) }}%
5860 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5861 \fi
5862 \endgroup}%
5863 \bbl@exp{%
5864 \bbl@ifunset{bbl@prehc@languagename}{}%
5865 {\bbl@ifblank{\bbl@cs{prehc@languagename}}{}}%
5866 {\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.

```

5867 \@onlypreamble\babelpatterns
5868 \AtEndOfPackage{%
5869 \newcommand\babelpatterns[2][\empty]{%
5870 \ifx\bbl@patterns\relax
5871 \let\bbl@patterns@\empty
5872 \fi
5873 \ifx\bbl@pttnlist\empty\else
5874 \bbl@warning{%
5875 You must not intermingle \string\selectlanguage\space and\%
5876 \string\babelpatterns\space or some patterns will not\%
5877 be taken into account. Reported}%
5878 \fi
5879 \ifx\@empty#1%
5880 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5881 \else
5882 \edef\bbl@tempb{\zap@space#1 \empty}%
5883 \bbl@for\bbl@tempa\bbl@tempb{%
5884 \bbl@fixname\bbl@tempa
5885 \bbl@iflanguage\bbl@tempa{%
5886 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5887 \@ifundefined{bbl@patterns@\bbl@tempa}%
5888 \empty
5889 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5890 #2}}}%
5891 \fi}}

```

11.6. Southeast Asian scripts

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

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

```

5892 \def\bbl@intraspace#1 #2 #3\@{%
5893 \directlua{
5894 Babel.intraspaces = Babel.intraspaces or {}
5895 Babel.intraspaces['\csname bbl@sbcp@languagename\endcsname'] = %
5896 {b = #1, p = #2, m = #3}
5897 Babel.locale_props[\the\localeid].intraspace = %
5898 {b = #1, p = #2, m = #3}
5899 }}
5900 \def\bbl@intrapenalty#1\@{%
5901 \directlua{

```

```

5902     Babel.intrapenalties = Babel.intrapenalties or {}
5903     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5904     Babel.locale_props[\the\localeid].intrapenalty = #1
5905   }}
5906 \begingroup
5907 \catcode\%=12
5908 \catcode\&=14
5909 \catcode\'=12
5910 \catcode\~=12
5911 \gdef\bbl@seaintraspace{&
5912   \let\bbl@seaintraspace\relax
5913   \directlua{
5914     Babel.sea_enabled = true
5915     Babel.sea_ranges = Babel.sea_ranges or {}
5916     function Babel.set_chrngs (script, chrng)
5917       local c = 0
5918       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5919         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5920         c = c + 1
5921       end
5922     end
5923     function Babel.sea_disc_to_space (head)
5924       local sea_ranges = Babel.sea_ranges
5925       local last_char = nil
5926       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5927       local item, next_item = head
5928       while item do
5929         next_item = item.next
5930         local i = item.id
5931         if i == node.id'glyph' then
5932           last_char = item
5933         elseif i == 7 and item.subtype == 3 and last_char
5934           and last_char.char > 0x0C99 then
5935           quad = font.getfont(last_char.font).size
5936           for lg, rg in pairs(sea_ranges) do
5937             if last_char.char > rg[1] and last_char.char < rg[2] then
5938               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5939               local intraspace = Babel.intraspaces[lg]
5940               local intrapenalty = Babel.intrapenalties[lg]
5941               local n
5942               if intrapenalty ~= 0 then
5943                 n = node.new(14, 0)      &% penalty
5944                 n.penalty = intrapenalty
5945                 node.insert_before(head, item, n)
5946               end
5947               n = node.new(12, 13)      &% (glue, spaceskip)
5948               node.setglue(n, intraspace.b * quad,
5949                 intraspace.p * quad,
5950                 intraspace.m * quad)
5951               node.insert_before(head, item, n)
5952               node.remove(head, item)
5953               node.free(item)
5954               break
5955             end
5956           end
5957           item = next_item
5958         end
5959       end
5960     end
5961   }&
5962   \bbl@luahyphenate}

```

11.7. 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.

```
5963 \catcode`\%=14
5964 \gdef\bbl@cjkintraspacespace{%
5965   \let\bbl@cjkintraspacespace\relax
5966   \directlua{
5967     require('babel-data-cjk.lua')
5968     Babel.cjk_enabled = true
5969     function Babel.cjk_linebreak(head)
5970       local GLYPH = node.id'glyph'
5971       local last_char = nil
5972       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5973       local last_class = nil
5974       local last_lang = nil
5975       for item in node.traverse(head) do
5976         if item.id == GLYPH then
5977           local lang = item.lang
5978           local LOCALE = node.get_attribute(item,
5979             Babel.attr_locale)
5980           local props = Babel.locale_props[LOCALE] or {}
5981           local class = Babel.cjk_class[item.char].c
5982           if props.cjk_quotes and props.cjk_quotes[item.char] then
5983             class = props.cjk_quotes[item.char]
5984           end
5985           if class == 'cp' then class = 'cl' % ) as CL
5986           elseif class == 'id' then class = 'I'
5987           elseif class == 'cj' then class = 'I' % loose
5988           end
5989           local br = 0
5990           if class and last_class and Babel.cjk_breaks[last_class][class] then
5991             br = Babel.cjk_breaks[last_class][class]
5992           end
5993           if br == 1 and props.linebreak == 'c' and
5994             lang ~= \the\l@nohyphenation\space and
5995             last_lang ~= \the\l@nohyphenation then
5996             local intrapenalty = props.intrapenalty
5997             if intrapenalty ~= 0 then
5998               local n = node.new(14, 0)      % penalty
5999               n.penalty = intrapenalty
6000               node.insert_before(head, item, n)
6001             end
6002             local intraspacespace = props.intraspacespace
6003             local n = node.new(12, 13)      % (glue, spaceskip)
6004             node.setglue(n, intraspacespace.b * quad,
6005               intraspacespace.p * quad,
6006               intraspacespace.m * quad)
6007             node.insert_before(head, item, n)
6008           end
6009           if font.getfont(item.font) then
6010             quad = font.getfont(item.font).size
6011           end
6012           last_class = class
6013           last_lang = lang
6014         else % if penalty, glue or anything else
6015           last_class = nil
6016         end
6017       end
end
```

```

6018     lang.hyphenate(head)
6019 end
6020 }%
6021 \bbl@luahyphenate}
6022 \gdef\bbl@luahyphenate{%
6023   \let\bbl@luahyphenate\relax
6024   \directlua{
6025     luatexbase.add_to_callback('hyphenate',
6026       function (head, tail)
6027         if Babel.protect_tail then
6028           Babel.protect_tail(head, tail)
6029         end
6030         if Babel.linebreaking.before then
6031           for k, func in ipairs(Babel.linebreaking.before) do
6032             func(head)
6033           end
6034         end
6035         lang.hyphenate(head)
6036         if Babel.cjk_enabled then
6037           Babel.cjk_linebreak(head)
6038         end
6039         if Babel.linebreaking.after then
6040           for k, func in ipairs(Babel.linebreaking.after) do
6041             func(head)
6042           end
6043         end
6044         if Babel.set_hboxed then
6045           Babel.set_hboxed(head)
6046         end
6047         if Babel.sea_enabled then
6048           Babel.sea_disc_to_space(head)
6049         end
6050       end,
6051       'Babel.hyphenate')
6052   }}
6053 \endgroup
6054 %
6055 \def\bbl@provide@intraspace{%
6056   \bbl@ifunset{\bbl@intsp@{language}}{%
6057     {\expandafter\ifx\cename\bbl@intsp@{language}\endcename\@empty\else
6058       \bbl@xin@{c}{\bbl@ccl{lnbrk}}}%
6059     \ifin@           % cjk
6060     \bbl@cjk@intraspace
6061     \directlua{
6062       Babel.locale_props = Babel.locale_props or {}
6063       Babel.locale_props[\the\localeid].linebreak = 'c'
6064     }%
6065     \bbl@exp{\bbl@intraspace\bbl@ccl{intsp}}\@{}%
6066     \ifx\bbl@KVP@intrapenalty\@nnil
6067       \bbl@intrapenalty0\@@
6068     \fi
6069   \else           % sea
6070     \bbl@sea@intraspace
6071     \bbl@exp{\bbl@intraspace\bbl@ccl{intsp}}\@{}%
6072     \directlua{
6073       Babel.sea_ranges = Babel.sea_ranges or {}
6074       Babel.set_chranges('\bbl@ccl{sbcpr}',
6075         '\bbl@ccl{chrng}')
6076     }%
6077     \ifx\bbl@KVP@intrapenalty\@nnil
6078       \bbl@intrapenalty0\@@
6079     \fi
6080   \fi

```

```

6081 \fi
6082 \ifx\bbl@KVP@intrapenalty\@nnil\else
6083 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
6084 \fi}}

```

11.8. Arabic justification

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

```

6085 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
6086 \def\bblar@chars{%
6087 0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
6088 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
6089 0640,0641,0642,0643,0644,0645,0646,0647,0649}
6090 \def\bblar@elongated{%
6091 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
6092 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
6093 0649,064A}
6094 \begingroup
6095 \catcode\_ =11 \catcode\`:=11
6096 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
6097 \endgroup
6098 \gdef\bbl@arabicjust{%
6099 \let\bbl@arabicjust\relax
6100 \newattribute\bblar@kashida
6101 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
6102 \bblar@kashida=\z@
6103 \bbl@patchfont{\bbl@parsejalt}}%
6104 \directlua{
6105 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
6106 Babel.arabic.elong_map[\the\localeid] = {}
6107 luatexbase.add_to_callback('post_linebreak_filter',
6108 Babel.arabic.justify, 'Babel.arabic.justify')
6109 luatexbase.add_to_callback('hpack_filter',
6110 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
6111 }%

```

Save both node lists to make replacement.

```

6112 \def\bblar@fetchjalt#1#2#3#4{%
6113 \bbl@exp{\bbl@foreach{#1}}{%
6114 \bbl@ifunset\bblar@JE@##1}%
6115 {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
6116 {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
6117 \directlua{%
6118 local last = nil
6119 for item in node.traverse(tex.box[0].head) do
6120 if item.id == node.id'glyph' and item.char > 0x600 and
6121 not (item.char == 0x200D) then
6122 last = item
6123 end
6124 end
6125 Babel.arabic.#3['##1#4'] = last.char
6126 }}}

```

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

```

6127 \gdef\bbl@parsejalt{%
6128 \ifx\addfontfeature\undefined\else
6129 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
6130 \ifin@
6131 \directlua{%
6132 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
6133 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}

```

```

6134         tex.print([[string\csname\space bbl@parsejalti\endcsname]])
6135     end
6136 }%
6137 \fi
6138 \fi}
6139 \gdef\bbl@parsejalti{%
6140 \begingroup
6141     \let\bbl@parsejalt\relax      % To avoid infinite loop
6142     \edef\bbl@tempb{\fontid\font}%
6143     \bblar@nofswarn
6144     \@nameuse{tracinglostchars}\z@
6145     \bblar@fetchjalt\bblar@elongated{}\{from\}\}%
6146     \bblar@fetchjalt\bblar@chars{^^^064a}\{from\}{a}% Alef maksura
6147     \bblar@fetchjalt\bblar@chars{^^^0649}\{from\}{y}% Yeh
6148     \addfontfeature{RawFeature=+jalt}%
6149     % \@namedef{\bblar@JE@0643}{06AA}% todo: catch medial kaf
6150     \bblar@fetchjalt\bblar@elongated{}\{dest\}\}%
6151     \bblar@fetchjalt\bblar@chars{^^^064a}\{dest\}{a}%
6152     \bblar@fetchjalt\bblar@chars{^^^0649}\{dest\}{y}%
6153     \directlua{%
6154         for k, v in pairs(Babel.arabic.from) do
6155             if Babel.arabic.dest[k] and
6156                 not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
6157                 Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
6158                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
6159             end
6160         end
6161     }%
6162 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

6163 \begingroup
6164 \catcode`#=11
6165 \catcode`~=11
6166 \directlua{
6167
6168 Babel.arabic = Babel.arabic or {}
6169 Babel.arabic.from = {}
6170 Babel.arabic.dest = {}
6171 Babel.arabic.justify_factor = 0.95
6172 Babel.arabic.justify_enabled = true
6173 Babel.arabic.kashida_limit = -1
6174
6175 function Babel.arabic.justify(head)
6176     if not Babel.arabic.justify_enabled then return head end
6177     local HLIST = node.id'hlist'
6178     local line, next_line = head
6179     while line do
6180         next_line = line.next
6181         if line.id == HLIST then
6182             % The line may be replaced (and freed), even the first one (eg,
6183             % in \vbox), so the head is returned.
6184             head = Babel.arabic.justify_hlist(head, line)
6185         end
6186         line = next_line
6187     end
6188     return head
6189 end
6190
6191 function Babel.arabic.justify_hbox(head, gc, size, pack)
6192     local has_inf = false
6193     if Babel.arabic.justify_enabled and pack == 'exactly' then
6194         for n in node.traverse_id(12, head) do

```

```

6195     if n.stretch_order > 0 then has_inf = true end
6196   end
6197   if not has_inf then
6198     head = Babel.arabic.justify_hlist(head, nil, gc, size, pack)
6199   end
6200 end
6201 return head
6202 end
6203
6204 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
6205   local d, new
6206   local k_list, k_item, pos_inline
6207   local width, width_new, full, k_curr, wt_pos, goal, shift
6208   local subst_done = false
6209   local elong_map = Babel.arabic.elong_map
6210   local cnt
6211   local last_line
6212   local GLYPH = node.id'glyph'
6213   local KASHIDA = Babel.attr_kashida
6214   local LOCALE = Babel.attr_locale
6215
6216   if line == nil then
6217     line = {}
6218     line.glue_sign = 1
6219     line.glue_order = 0
6220     line.head = head
6221     line.shift = 0
6222     line.width = size
6223   end
6224
6225   % Exclude last line.
6226   if ((line.next ~= nil or line.glue_sign == 1) and line.glue_order == 0) then
6227     elongs = {}      % Stores elongated candidates of each line
6228     k_list = {}      % And all letters with kashida
6229     pos_inline = 0   % Not yet used
6230
6231     for n in node.traverse_id(GLYPH, line.head) do
6232       pos_inline = pos_inline + 1 % To find where it is. Not used.
6233
6234       % Elongated glyphs
6235       if elong_map then
6236         local locale = node.get_attribute(n, LOCALE)
6237         if elong_map[locale] and elong_map[locale][n.font] and
6238           elong_map[locale][n.font][n.char] then
6239           table.insert(elongs, {node = n, locale = locale} )
6240           node.set_attribute(n.prev, KASHIDA, 0)
6241         end
6242       end
6243
6244       % Tatwil. First create a list of nodes marked with kashida. The
6245       % rest of nodes can be ignored. The list of used weights is build
6246       % when transforms with the key kashida= are declared.
6247       if Babel.kashida_wts then
6248         local k_wt = node.get_attribute(n, KASHIDA)
6249         if k_wt > 0 then % todo. parameter for multi inserts
6250           table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
6251         end
6252       end
6253
6254     end % of node.traverse_id
6255
6256     if #elongs == 0 and #k_list == 0 then goto next_line end
6257     full = line.width

```

```

6258     shift = line.shift
6259     goal = full * Babel.arabic.justify_factor % A bit crude
6260     width = node.dimensions(line.head) % The 'natural' width
6261
6262     % == Elongated ==
6263     % Original idea taken from 'chickenize'
6264     while (#elongs > 0 and width < goal) do
6265         subst_done = true
6266         local x = #elongs
6267         local curr = elongs[x].node
6268         local oldchar = curr.char
6269         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6270         width = node.dimensions(line.head) % Check if the line is too wide
6271         % Substitute back if the line would be too wide and break:
6272         if width > goal then
6273             curr.char = oldchar
6274             break
6275         end
6276         % If continue, pop the just substituted node from the list:
6277         table.remove(elongs, x)
6278     end
6279
6280     % == Tatwil ==
6281     % Traverse the kashida node list so many times as required, until
6282     % the line is filled. The first pass adds a tatweel after each
6283     % node with kashida in the line, the second pass adds another one,
6284     % and so on. In each pass, add first the kashida with the highest
6285     % weight, then with lower weight and so on.
6286     if #k_list == 0 then goto next_line end
6287
6288     width = node.dimensions(line.head) % The 'natural' width
6289     k_curr = #k_list % Traverse backwards, from the end
6290     wt_pos = 1
6291
6292     while width < goal do
6293         subst_done = true
6294         k_item = k_list[k_curr].node
6295         if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6296             d = node.copy(k_item)
6297             d.char = 0x0640
6298             d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6299             d.xoffset = 0
6300             line.head, new = node.insert_after(line.head, k_item, d)
6301             width_new = node.dimensions(line.head)
6302             if width > goal or width == width_new then
6303                 node.remove(line.head, new) % Better compute before
6304                 node.free(new)
6305                 break
6306             end
6307             if Babel.fix_diacr then
6308                 Babel.fix_diacr(k_item.next)
6309             end
6310             width = width_new
6311         end
6312         if k_curr == 1 then
6313             k_curr = #k_list
6314             wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6315         else
6316             k_curr = k_curr - 1
6317         end
6318     end
6319
6320     % Limit the number of tatweel by removing them. Not very efficient,

```

```

6321 % but it does the job in a quite predictable way.
6322 if Babel.arabic.kashida_limit > -1 then
6323   cnt = 0
6324   local n, next_n = line.head
6325   while n do
6326     next_n = n.next
6327     if n.id == GLYPH and n.char == 0x0640 then
6328       cnt = cnt + 1
6329       if cnt > Babel.arabic.kashida_limit then
6330         line.head = node.remove(line.head, n)
6331         node.free(n)
6332       end
6333     elseif n.id == GLYPH then
6334       cnt = 0
6335     end
6336     n = next_n
6337   end
6338 end
6339
6340 ::next_line::
6341
6342 % Must take into account marks and ins, see luatex manual.
6343 % Have to be executed only if there are changes. Investigate
6344 % what's going on exactly.
6345 if subst_done and not gc then
6346   d = node.hpack(line.head, full, 'exactly')
6347   d.shift = shift
6348   head = node.insert_before(head, line, d)
6349   head = node.remove(head, line)
6350   line.head = nil % Now in d
6351   node.free(line)
6352 end
6353 end % if process line
6354 % The list of a box (line is a table) may start with a removed tatweel
6355 return (line.id and head) or line.head
6356 end
6357 }
6358 \endgroup
6359 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

11.9. Common stuff

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

```

6360 \def\bbl@scr@node@list{%
6361   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6362   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6363 \ifnum\bbl@bidimode=102 % bidi-r
6364   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6365 \fi
6366 \def\bbl@set@renderer{%
6367   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6368   \ifin@
6369     \let\bbl@unset@renderer\relax
6370   \else
6371     \bbl@exp{%
6372       \def\\bbl@unset@renderer{%
6373         \def<g__fontspec_default_fontopts_clist>{%
6374           [g__fontspec_default_fontopts_clist]}%
6375         \def<g__fontspec_default_fontopts_clist>{%

```

```

6376         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
6377     \fi}
6378 <@Font selection@>

```

11.10 Automatic fonts and ids switching

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

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

```

6379 \directlua{% DL6
6380 Babel.script_blocks = {
6381   ['dflt'] = {},
6382   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6383               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6384   ['Armn'] = {{0x0530, 0x058F}},
6385   ['Beng'] = {{0x0980, 0x09FF}},
6386   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6387   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6388   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6389              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6390   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6391   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6392              {0xAB00, 0xAB2F}},
6393   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6394   % Don't follow strictly Unicode, which places some Coptic letters in
6395   % the 'Greek and Coptic' block
6396   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6397   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6398              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6399              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6400              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6401              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6402              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6403   ['Hebr'] = {{0x0590, 0x05FF},
6404              {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6405   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6406              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6407   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6408   ['Knda'] = {{0x0C80, 0x0CFF}},
6409   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6410              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6411              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6412   ['Lao'] = {{0x0E80, 0x0EFF}},
6413   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6414              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6415              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6416   ['Mahj'] = {{0x11150, 0x1117F}},
6417   ['Mlym'] = {{0x0D00, 0x0D7F}},
6418   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6419   ['Orya'] = {{0x0B00, 0x0B7F}},
6420   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6421   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6422   ['Taml'] = {{0x0B80, 0x0BFF}},
6423   ['Telu'] = {{0x0C00, 0x0C7F}},
6424   ['Tfng'] = {{0x2D30, 0x2D7F}},

```

```

6425 ['Thai'] = {{0x0E00, 0x0E7F}},
6426 ['Tibt'] = {{0x0F00, 0x0FFF}},
6427 ['Vaii'] = {{0xA500, 0xA63F}},
6428 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6429 }
6430
6431 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6432 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6433 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6434
6435 function Babel.locale_map(head)
6436   if not Babel.locale_mapped then return head end
6437
6438   local LOCALE = Babel.attr_locale
6439   local GLYPH = node.id('glyph')
6440   local inmath = false
6441   local toloc_save
6442   for item in node.traverse(head) do
6443     local toloc
6444     if not inmath and item.id == GLYPH then
6445       % Optimization: build a table with the chars found
6446       if Babel.chr_to_loc[item.char] then
6447         toloc = Babel.chr_to_loc[item.char]
6448       else
6449         for lc, maps in pairs(Babel.loc_to_scr) do
6450           for _, rg in pairs(maps) do
6451             if item.char >= rg[1] and item.char <= rg[2] then
6452               Babel.chr_to_loc[item.char] = lc
6453               toloc = lc
6454               break
6455             end
6456           end
6457         end
6458         % Treat composite chars in a different fashion, because they
6459         % 'inherit' the previous locale.
6460         if (item.char >= 0x0300 and item.char <= 0x036F) or
6461            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6462            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6463           Babel.chr_to_loc[item.char] = -2000
6464           toloc = -2000
6465         end
6466         if not toloc then
6467           Babel.chr_to_loc[item.char] = -1000
6468         end
6469       end
6470       if toloc == -2000 then
6471         toloc = toloc_save
6472       elseif toloc == -1000 then
6473         toloc = nil
6474       end
6475       if toloc and Babel.locale_props[toloc] and
6476          Babel.locale_props[toloc].letters and
6477          tex.getcatcode(item.char) \string~= 11 then
6478         toloc = nil
6479       end
6480       if toloc and Babel.locale_props[toloc].script
6481          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6482          and Babel.locale_props[toloc].script ==
6483          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6484         toloc = nil
6485       end
6486       if toloc then
6487         if Babel.locale_props[toloc].lg then

```

```

6488         item.lang = Babel.locale_props[toloc].lg
6489         node.set_attribute(item, LOCALE, toloc)
6490     end
6491     if Babel.locale_props[toloc]['/'..item.font] then
6492         item.font = Babel.locale_props[toloc]['/'..item.font]
6493     end
6494 end
6495 toloc_save = toloc
6496 elseif not inmath and item.id == 7 then % Apply recursively
6497     item.replace = item.replace and Babel.locale_map(item.replace)
6498     item.pre      = item.pre and Babel.locale_map(item.pre)
6499     item.post     = item.post and Babel.locale_map(item.post)
6500 elseif item.id == node.id'math' then
6501     inmath = (item.subtype == 0)
6502 end
6503 end
6504 return head
6505 end
6506 }

```

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

```

6507 \newcommand\babelcharproperty[1]{%
6508   \count@=#1\relax
6509   \ifvmode
6510     \expandafter\bbl@chprop
6511   \else
6512     \bbl@error{charproperty-only-vertical}{}{}{}%
6513   \fi}
6514 \newcommand\bbl@chprop[3][\the\count@]{%
6515   \@tempcnta=#1\relax
6516   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6517   {\bbl@error{unknown-char-property}{}{#2}{}%
6518   {}%
6519   \loop
6520     \bbl@cs{chprop@#2}{#3}%
6521     \ifnum\count@<\@tempcnta
6522       \advance\count@\@ne
6523     \repeat}
6524 %
6525 \def\bbl@chprop@direction#1{%
6526   \directlua{
6527     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6528     Babel.characters[\the\count@]['d'] = '#1'
6529   }}
6530 \let\bbl@chprop@bc\bbl@chprop@direction
6531 %
6532 \def\bbl@chprop@mirror#1{%
6533   \directlua{
6534     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6535     Babel.characters[\the\count@]['m'] = '\number#1'
6536   }}
6537 \let\bbl@chprop@bmg\bbl@chprop@mirror
6538 %
6539 \def\bbl@chprop@linebreak#1{%
6540   \directlua{
6541     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6542     Babel.cjk_characters[\the\count@]['c'] = '#1'
6543   }}
6544 \let\bbl@chprop@lb\bbl@chprop@linebreak
6545 %
6546 \def\bbl@chprop@locale#1{%
6547   \directlua{

```

```

6548   Babel.chr_to_loc = Babel.chr_to_loc or {}
6549   Babel.chr_to_loc[\the\count@] =
6550     \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6551   }}

```

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.

```

6552 \directlua{% DL7
6553   Babel.nohyphenation = \the\l@nohyphenation
6554 }

```

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

```

6555 \beginingroup
6556 \catcode\~ = 12
6557 \catcode\% = 12
6558 \catcode\& = 14
6559 \catcode\| = 12
6560 \gdef\babelprehyphenation{%&
6561   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6562 \gdef\babelposthyphenation{%&
6563   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6564 %
6565 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6566   \ifcase#1
6567     \bbl@activateprehyphen
6568   \or
6569     \bbl@activateposthyphen
6570   \fi
6571 \beginingroup
6572   \def\babeltempa{\bbl@add@list\babeltempb}%&
6573   \let\babeltempb\empty
6574   \def\bbl@tempa{#5}%&
6575   \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6576   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6577     \bbl@ifsamestring{##1}{remove}%&
6578     {\bbl@add@list\babeltempb{nil}}}%&
6579   {\directlua{
6580     local rep = {[#1]}=
6581     local three_args = '%s*=%s*([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)'
6582     & Numeric passes directly: kern, penalty...
6583     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6584     rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
6585     rep = rep:gsub('^%s*(after)%s*$', 'after = true, ')
6586     rep = rep:gsub('(string)%s*=%s*([%^s,]*)', Babel.capture_func)
6587     rep = rep:gsub('node%s*=%s*([%a+)%s*([%a*])', Babel.capture_node)
6588     rep = rep:gsub(' (norule)' .. three_args,
6589       'norule = {' .. '%2, %3, %4' .. '}')
6590     if #1 == 0 or #1 == 2 then
6591       rep = rep:gsub(' (space)' .. three_args,
6592         'space = {' .. '%2, %3, %4' .. '}')
6593       rep = rep:gsub(' (spacefactor)' .. three_args,
6594         'spacefactor = {' .. '%2, %3, %4' .. '}')
6595       rep = rep:gsub(' (kashida)%s*=%s*([%^s,]*)', Babel.capture_kashida)
6596       & Transform values
6597       rep, n = rep:gsub(' ({[%a%-%.]|([%a%_%.]|+))',
6598         function(v,d)

```

```

6599         return string.format (
6600             '{\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6601             v,
6602             load( 'return Babel.locale_props'..
6603                 '[\the\csname bbl@id@@#3\endcsname].' .. d)() )
6604         end )
6605         rep, n = rep:gsub( '{([%a%-%.]+)|([%-%d%.]+)}',
6606             '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6607     end
6608     if #1 == 1 then
6609         rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6610         rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6611         rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6612     end
6613     tex.print([[\\string\babeltempa{[]] .. rep .. [[]]])
6614 }}&%
6615 \bbl@foreach\babeltempb{&%
6616     \bbl@forkv{##1}{&%
6617         \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6618             post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6619         \ifin@else
6620             \bbl@error{bad-transform-option}{###1}{,}{&%
6621                 \fi}}&%
6622 \let\bbl@kv@attribute\relax
6623 \let\bbl@kv@label\relax
6624 \let\bbl@kv@fonts\empty
6625 \let\bbl@kv@prepend\relax
6626 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6627 \ifx\bbl@kv@fonts\empty\else\bbl@settransfont\fi
6628 \ifx\bbl@kv@attribute\relax
6629     \ifx\bbl@kv@label\relax\else
6630         \bbl@exp{\\bbl@trim@def\\bbl@kv@fonts{\bbl@kv@fonts}}&%
6631         \bbl@replace\bbl@kv@fonts{ }{,}&%
6632         \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6633         \count@\z@
6634         \def\bbl@elt##1##2##3{&%
6635             \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6636             {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6637                 {\count@\@ne}&%
6638                 {\bbl@error{font-conflict-transforms}{,}{,}}}&%
6639             }&%
6640         \bbl@transfont@list
6641         \ifnum\count@=\z@
6642             \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6643                 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6644             \fi
6645             \bbl@ifunset{\bbl@kv@attribute}&%
6646             {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6647             {}&%
6648             \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6649             \fi
6650     \else
6651         \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6652         \fi
6653     \directlua{
6654         local lbr = Babel.linebreaking.replacements[#1]
6655         local u = unicode.utf8
6656         local id, attr, label
6657         if #1 == 0 then
6658             id = \the\csname bbl@id@@#3\endcsname\space
6659         else
6660             id = \the\csname l@#3\endcsname\space
6661         end

```

```

6662 \ifx\bbl@kv@attribute\relax
6663 attr = -1
6664 \else
6665 attr = luatexbase.registernumber'\bbl@kv@attribute'
6666 \fi
6667 \ifx\bbl@kv@label\relax\else &% Same refs:
6668 label = [==[\bbl@kv@label]==]
6669 \fi
6670 &% Convert pattern:
6671 local patt = string.gsub([==[#4]==], '%s', '')
6672 if #1 == 0 then
6673 patt = string.gsub(patt, '|', ' ')
6674 end
6675 if not u.find(patt, '()', nil, true) then
6676 patt = '()' .. patt .. '()'
6677 end
6678 patt = string.gsub(patt, '%(%)%', '^()')
6679 patt = string.gsub(patt, '%$$(%)', '()$')
6680 patt = u.gsub(patt, '{(.)}',
6681 function (n)
6682 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6683 end)
6684 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6685 function (n)
6686 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6687 end)
6688 lbkr[id] = lbkr[id] or {}
6689 table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6690 { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6691 }&%
6692 \endgroup}
6693 \endgroup
6694 %
6695 \let\bbl@transfont@list\@empty
6696 \def\bbl@settransfont{%
6697 \global\let\bbl@settransfont\relax % Execute only once
6698 \gdef\bbl@transfont{%
6699 \def\bbl@elt####1####2####3{%
6700 \bbl@ifblank{####3}%
6701 {\count@tw@}% Do nothing if no fonts
6702 {\count@z@
6703 \bbl@vforeach{####3}{%
6704 \def\bbl@tempd{#####1}%
6705 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6706 \ifx\bbl@tempd\bbl@tempe
6707 \count@ne
6708 \else\ifx\bbl@tempd\bbl@transfam
6709 \count@ne
6710 \fi\fi}%
6711 \ifcase\count@
6712 \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
6713 \or
6714 \bbl@csarg\setattribute{ATR####2@####1@####3}\@ne
6715 \fi}}%
6716 \bbl@transfont@list}%
6717 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6718 \gdef\bbl@transfam{-unknown-}%
6719 \bbl@foreach\bbl@font@fams{%
6720 \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6721 \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6722 {\xdef\bbl@transfam{##1}}%
6723 {}}}
6724 %

```

```

6725 \DeclareRobustCommand\enablelocaletransform[1]{%
6726   \bbl@ifunset{bbl@ATR@#1@\language @}{%
6727     {\bbl@error{transform-not-available}{#1}}{}}%
6728     {\bbl@csarg\setattribute{ATR@#1@\language @}{\@ne}}
6729 \DeclareRobustCommand\disablelocaletransform[1]{%
6730   \bbl@ifunset{bbl@ATR@#1@\language @}{%
6731     {\bbl@error{transform-not-available-b}{#1}}{}}%
6732     {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}

```

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

```

6733 \def\bbl@activateposthyphen{%
6734   \let\bbl@activateposthyphen\relax
6735   \ifx\bbl@attr@hboxed\@undefined
6736     \newattribute\bbl@attr@hboxed
6737   \fi
6738   \directlua{
6739     require('babel-transforms.lua')
6740     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6741   }}
6742 \def\bbl@activateprehyphen{%
6743   \let\bbl@activateprehyphen\relax
6744   \ifx\bbl@attr@hboxed\@undefined
6745     \newattribute\bbl@attr@hboxed
6746   \fi
6747   \directlua{
6748     require('babel-transforms.lua')
6749     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6750   }}
6751 \newcommand\SetTransformValue[3]{%
6752   \directlua{
6753     Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6754   }}

```

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

```

6755 \newcommand\ShowBabelTransforms[1]{%
6756   \bbl@activateprehyphen
6757   \bbl@activateposthyphen
6758   \begingroup
6759     \directlua{ Babel.show_transforms = true }%
6760     \setbox\z@\vbox{#1}%
6761     \directlua{ Babel.show_transforms = false }%
6762   \endgroup}

```

The following experimental (and unfinished) macro applies the prehyphenation transforms for the current locale to a string (characters and spaces) and processes it in a fully expandable way (among other limitations, the string can't contain `]==]`). The way it operates is admittedly rather cumbersome: it converts the string to a node list, processes it, and converts it back to a string. The lua code is in the lua file below.

```

6763 \newcommand\localeprehyphenation[1]{%
6764   \directlua{ Babel.string_prehyphenation([==#1==], \the\localeid) }}

```

11.11.Bidi

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

```

6765 \AtBeginDocument{\pagedir\bodydir}

```

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

```

6766 \def\bbl@activate@preotf{%
6767   \let\bbl@activate@preotf\relax % only once
6768   \directlua{
6769     function Babel.pre_otfload_v(head)
6770       if Babel.numbers and Babel.digits_mapped then
6771         head = Babel.numbers(head)
6772       end
6773       if Babel.bidi_enabled then
6774         head = Babel.bidi(head, false, dir)
6775       end
6776       return head
6777     end
6778   %
6779   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6780     if Babel.numbers and Babel.digits_mapped then
6781       head = Babel.numbers(head)
6782     end
6783     if Babel.bidi_enabled then
6784       head = Babel.bidi(head, false, dir)
6785     end
6786     return head
6787   end
6788   %
6789   luatexbase.add_to_callback('pre_linebreak_filter',
6790     Babel.pre_otfload_v,
6791     'Babel.pre_otfload_v',
6792     Babel.priority_in_callback('pre_linebreak_filter',
6793       'luaotfload.node_processor') or nil)
6794   %
6795   luatexbase.add_to_callback('hpack_filter',
6796     Babel.pre_otfload_h,
6797     'Babel.pre_otfload_h',
6798     Babel.priority_in_callback('hpack_filter',
6799       'luaotfload.node_processor') or nil)
6800 }

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@insidemath` hack is activated (set to 1) every math with the package option `bidi=`. The hack for the PUA is no longer necessary with basic (24.8), but it's kept in basic-r.

```

6801 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6802   \let\bbl@beforeforeign\leavevmode
6803   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6804 \ifnum\bbl@bidimode>300
6805   \RequirePackage{unibidi-lua}
6806 \else
6807   \RequirePackage{luatexbase}
6808   \bbl@activate@preotf
6809   \directlua{
6810     require('babel-data-bidi.lua')
6811     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6812       require('babel-bidi-basic.lua')
6813     \or
6814       require('babel-bidi-basic-r.lua')
6815       table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6816       table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6817       table.insert(Babel.ranges, {0x10000, 0x10FFFFD, 'on'})
6818     \fi
6819   \fi
6820   \newattribute\bbl@attr@dir
6821   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6822   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6823 \fi

```

```

6824 %
6825 \chardef\bbl@thetextdir\z@
6826 \chardef\bbl@thepardir\z@
6827 \def\bbl@setluadir#1#2{% 1=\text/pardirection 2= 0:l 1:r 2:a\l
6828 \ifcase#2\relax
6829 \ifcase#1\else#1=\z@\fi
6830 \else
6831 \ifcase#1#1=\@ne\fi
6832 \fi}

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

```

```

6833 \def\bbl@thedir{0}
6834 \def\bbl@textdir#1{%
6835 \bbl@setluadir\textdirection{#1}%
6836 \chardef\bbl@thetextdir#1\relax
6837 \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6838 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6839 \def\bbl@pardir#1{% Used twice
6840 \bbl@setluadir\pardirection{#1}%
6841 \chardef\bbl@thepardir#1\relax}
6842 \def\bbl@bodydir{\bbl@setluadir\bodydirection}% Used once
6843 \def\bbl@dirparastext{\pardirection=\textdirection\relax}% Used once

```

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

```

6844 \ifnum\bbl@bidimode>\z@ % Any bidi=
6845 \def\bbl@insidemath{0}%
6846 \def\bbl@everymath{\def\bbl@insidemath{1}}
6847 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6848 \frozen@everymath\expandafter{%
6849 \expandafter\bbl@everymath\the\frozen@everymath}
6850 \frozen@everydisplay\expandafter{%
6851 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6852 \AtBeginDocument{
6853 \directlua{
6854 function Babel.math_box_dir(head)
6855 if not (token.get_macro('bbl@insidemath') == '0') then
6856 if Babel.hlist_has_bidi(head) then
6857 local d = node.new(node.id'dir')
6858 d.dir = '+TRT'
6859 for item in node.traverse(head) do
6860 if item.id == 11 or item.id == node.id'glyph' then
6861 head = node.insert_before(head, item, d)
6862 break
6863 end
6864 end
6865 local inmath = false
6866 for item in node.traverse(head) do
6867 if item.id == 11 then
6868 inmath = (item.subtype == 0)
6869 elseif not inmath then
6870 node.set_attribute(item,
6871 Babel.attr_dir, token.get_macro('bbl@thedir'))
6872 end
6873 end
6874 end
6875 end
6876 return head
6877 end
6878 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6879 "Babel.math_box_dir", 0)
6880 if Babel.unset_atdir then
6881 luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,

```

```

6882         "Babel.unset_atdir")
6883         luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6884         "Babel.unset_atdir")
6885     end
6886     luatexbase.add_to_callback("post_mlist_to_hlist_filter", function(head)
6887         local dir = tex.mathdirection
6888         local n = node.new("dir")
6889         n.direction = dir
6890         head = node.insert_before(head, head, n)
6891         n = node.new("dir", 1)
6892         n.direction = dir
6893         head = node.insert_after(head, node.tail(head), n)
6894         return head
6895     end, "Babel.ensure_math_dir")
6896 } }%
6897 \fi
6898 \ifnum\bbbl@bidimode>300
6899   \def\bbbl@sublr#1{^^^2066#1^^^2069}
6900 \fi

Experimental. Tentative name.

6901 \DeclareRobustCommand\localebox[1]{%
6902   {\def\bbbl@insidemath{0}%
6903     \mbox{\foreignlanguage{\language}\{#1}}}}

```

11.12 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 `bidimode=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.

```

6904 \bbbl@trace{Redefinitions for bidi layout}
6905 %
6906 ({*More package options}) ≡
6907 \chardef\bbbl@eqnpos\z@
6908 \DeclareOption{leqno}{\chardef\bbbl@eqnpos\@ne}
6909 \DeclareOption{fleqn}{\chardef\bbbl@eqnpos\tw@}
6910 ({/More package options})

```

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

```

6911 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6912   \matheqdirmode\@ne
6913   \mathemptydisplaymode\@ne
6914   \breakafterdirmode\@ne
6915   \fixupboxesmode\@ne
6916   \let\bbbl@eqnudir\relax
6917   \def\bbbl@eqdel{()}
6918   \def\bbbl@eqnum{%

```

```

6919     {\normalfont\normalcolor
6920     \expandafter\@firstoftwo\bbl@eqdel
6921     \theequation
6922     \expandafter\@secondoftwo\bbl@eqdel}}
6923 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6924 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6925 \def\bbl@eqno@flip#1{% So
6926     \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6927         \eqno
6928         \hb@xt@.01pt{%
6929             \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6930     \else
6931         \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6932     \fi
6933     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6934 \def\bbl@leqno@flip#1{%
6935     \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6936         \leqno
6937         \hb@xt@.01pt{%
6938             \hss\hb@xt@\displaywidth{{#1}\glet\bbl@upset\@currentlabel}\hss}}%
6939     \else
6940         \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6941     \fi
6942     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6943 %
6944 \AtBeginDocument{%
6945     \ifx\bbl@noamsmath\relax\else
6946     \ifx\maketag@@@ \@undefined % Normal equation, eqnarray
6947         \ifnum\bbl@eqnpos=\tw@
6948             \bbl@replace\equation{\hb@xt@\linewidth}%
6949             {\hbox bdir\mathdirection to\linewidth}%
6950             \bbl@carg\bbl@sreplace{[ ]}{\hb@xt@\linewidth}%
6951             {\hbox bdir\mathdirection to\linewidth}%
6952         \fi
6953         \AddToHook{env/equation/begin}{%
6954             \ifnum\bbl@thetextdir>\z@
6955                 \let\@eqnnum\bbl@eqnum
6956                 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6957                 \chardef\bbl@thetextdir\z@
6958                 \bbl@add\normalfont{\bbl@eqnodir}%
6959                 \ifcase\bbl@eqnpos
6960                     \let\bbl@puteqno\bbl@eqno@flip
6961                 \or
6962                     \let\bbl@puteqno\bbl@leqno@flip
6963                 \fi
6964             \fi}%
6965         \ifnum\bbl@eqnpos=\tw@\else
6966             \def\endequation{\bbl@puteqno{\@eqnnum}$\@ignoretrue}%
6967         \fi
6968         \AddToHook{env/eqnarray/begin}{%
6969             \ifnum\bbl@thetextdir>\z@
6970                 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6971                 \chardef\bbl@thetextdir\z@
6972                 \bbl@add\normalfont{\bbl@eqnodir}%
6973                 \ifnum\bbl@eqnpos=\@ne
6974                     \def\@eqnnum{%
6975                         \setbox\z@\hbox{\bbl@eqnum}%
6976                         \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6977                 \else
6978                     \let\@eqnnum\bbl@eqnum
6979                 \fi
6980             \fi}
6981     \else % amstex

```

```

6982 \bbl@exp{% Hack to hide maybe undefined conditionals:
6983 \chardef\bbl@eqnpos=0\<iftagsleft>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6984 \ifnum\bbl@eqnpos=\@ne
6985 \let\bbl@ams@lap\hbox
6986 \else
6987 \let\bbl@ams@lap\llap
6988 \fi
6989 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6990 \bbl@sreplace\intertext@{\normalbaselines}%
6991 {\normalbaselines
6992 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6993 \ExplSyntaxOff
6994 \def\bbl@ams@tagbox#1#2{\#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6995 \ifx\bbl@ams@lap\hbox % leqno
6996 \def\bbl@ams@flip#1{%
6997 \hbox to 0.01pt{\hss\hbox to\displaywidth{\#1}\hss}}%
6998 \else % eqno
6999 \def\bbl@ams@flip#1{%
7000 \hbox to 0.01pt{\hbox to\displaywidth{\hss{\#1}}\hss}}%
7001 \fi
7002 \def\bbl@ams@preset#1{%
7003 \ifnum\bbl@thetextdir>\z@
7004 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
7005 \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
7006 \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
7007 \fi}%
7008 \def\bbl@ams@equation{%
7009 \ifnum\bbl@thetextdir>\z@
7010 \bbl@ams@preset\hbox
7011 \chardef\bbl@thetextdir\z@
7012 \bbl@add\normalfont{\bbl@eqnodir}%
7013 \ifcase\bbl@eqnpos
7014 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
7015 \or
7016 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
7017 \fi
7018 \fi}%
7019 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
7020 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
7021 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
7022 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
7023 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
7024 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
7025 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
7026 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
7027 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
7028 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
7029 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
7030 % Hackish, for proper alignment. Don't ask me why it works!:
7031 \bbl@exp{% Avoid a 'visible' conditional
7032 \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
7033 \\\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
7034 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
7035 \AddToHook{env/split/before}{%
7036 \ifnum\bbl@thetextdir>\z@
7037 \bbl@ifsamestring\@currentenv{equation}%
7038 {\ifx\bbl@ams@lap\hbox % leqno
7039 \def\bbl@ams@flip#1{%
7040 \hbox to 0.01pt{\hbox to\displaywidth{\#1}\hss}\hss}}%
7041 \else
7042 \def\bbl@ams@flip#1{%
7043 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{\#1}}}%
7044 \fi}%

```

```

7045         {}%
7046     \fi}%
7047 \fi\fi}
7048 \fi

Declarations specific to lua, called by \babelprovide.

7049 \def\bbbl@provide@extra#1{%
7050     % == onchar ==
7051     \ifx\bbbl@KVP@onchar\@nnil\else
7052         \bbbl@luahyphenate
7053         \bbbl@exp{%
7054             \\AddToHook{env/document/before}{%
7055                 {\let\\bbbl@ifrestoring\\@firstoftwo
7056                     \\select@language{#1}{}}}%
7057         \directlua{
7058             if Babel.locale_mapped == nil then
7059                 Babel.locale_mapped = true
7060                 Babel.linebreaking.add_before(Babel.locale_map, 1)
7061                 Babel.loc_to_scr = {}
7062                 Babel.chr_to_loc = Babel.chr_to_loc or {}
7063             end
7064             Babel.locale_props[\the\localeid].letters = false
7065         }%
7066         \bbbl@xin@{ letters }{ \bbbl@KVP@onchar\space}%
7067         \ifin@
7068             \directlua{
7069                 Babel.locale_props[\the\localeid].letters = true
7070             }%
7071         \fi
7072         \bbbl@xin@{ ids }{ \bbbl@KVP@onchar\space}%
7073         \ifin@
7074             \ifx\bbbl@starthyphens\@undefined % Needed if no explicit selection
7075                 \AddBabelHook{babel-onchar}{beforestart}{\bbbl@starthyphens}%
7076             \fi
7077             \bbbl@exp{\\bbbl@add\\bbbl@starthyphens
7078                 {\bbbl@patterns@lua{\language#1}}}%
7079             \directlua{
7080                 if Babel.script_blocks['\bbbl@cl{sbcpr}'] then
7081                     Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbbl@cl{sbcpr}']
7082                     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language#1}\space
7083                 end
7084             }%
7085         \fi
7086         \bbbl@xin@{ fonts }{ \bbbl@KVP@onchar\space}%
7087         \ifin@
7088             \bbbl@ifunset{bbbl@lsys@\language#1}{\bbbl@provide@lsys{\language#1}}{%
7089             \bbbl@ifunset{bbbl@wdir@\language#1}{\bbbl@provide@dirs{\language#1}}{%
7090             \directlua{
7091                 if Babel.script_blocks['\bbbl@cl{sbcpr}'] then
7092                     Babel.loc_to_scr[\the\localeid] =
7093                     Babel.script_blocks['\bbbl@cl{sbcpr}']
7094                 end}%
7095             \ifx\bbbl@mapselect\@undefined
7096                 \AtBeginDocument{%
7097                     \bbbl@patchfont{\bbbl@mapselect}%
7098                     {\selectfont}}%
7099                 \def\bbbl@mapselect{%
7100                     \let\bbbl@mapselect\relax
7101                     \edef\bbbl@prefontid{\fontid\font}}%
7102                 \def\bbbl@mapdir##1{%
7103                     \begingroup
7104                     \setbox\z@\hbox{% Force text mode
7105                     \def\language#1}%

```

```

7106         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
7107         \bbl@switchfont
7108         \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
7109         \directlua{
7110             Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
7111             ['/bbl@prefontid'] = \fontid\font\space}%
7112         \fi}%
7113     \endgroup}%
7114     \fi
7115     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7116     \fi
7117 \fi
7118 % == mapfont ==
7119 % For bidi texts, to switch the font based on direction. Deprecated
7120 \ifx\bbl@KVP@mapfont\@nnil\else
7121     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{%
7122         {\bbl@error{unknown-mapfont}}{}}}%
7123     \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys{\language}}{%
7124         \bbl@ifunset{\bbl@wdir\language}{\bbl@provide@dirs{\language}}{%
7125             \ifx\bbl@mapselect\@undefined
7126                 \AtBeginDocument{%
7127                     \bbl@patchfont{\bbl@mapselect}}%
7128                     {\selectfont}}%
7129                 \def\bbl@mapselect{%
7130                     \let\bbl@mapselect\relax
7131                     \edef\bbl@prefontid{\fontid\font}}%
7132                 \def\bbl@mapdir##1{%
7133                     {\def\language{##1}%
7134                     \let\bbl@ifrestoring\@firstoftwo % avoid font warning
7135                     \bbl@switchfont
7136                     \directlua{Babel.fontmap
7137                     [\the\csname bbl@wdir@##1\endcsname]%
7138                     [\bbl@prefontid]=\fontid\font}}}%
7139                 \fi
7140                 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7141             \fi
7142             % == Line breaking: CJK quotes ==
7143             \ifcase\bbl@engine\or
7144                 \bbl@xin{/c}{\bbl@cl{\lnbrk}}%
7145             \ifin@
7146                 \bbl@ifunset{\bbl@quote\language}{%
7147                     {\directlua{
7148                         Babel.locale_props[\the\localeid].cjk_quotes = {}
7149                         local cs = 'op'
7150                         for c in string.utfvalues(%
7151                             [[\csname bbl@quote\language\endcsname]]) do
7152                             if Babel.cjk_characters[c].c == 'qu' then
7153                                 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
7154                             end
7155                             cs = ( cs == 'op') and 'cl' or 'op'
7156                             end
7157                         }}%
7158                     \fi
7159                 \fi
7160             % == Counters: mapdigits ==
7161             % Native digits
7162             \ifx\bbl@KVP@mapdigits\@nnil\else
7163                 \bbl@ifunset{\bbl@dgnat\language}{%
7164                     {\bbl@activate@preotf
7165                     \directlua{
7166                         Babel.digits_mapped = true
7167                         Babel.digits = Babel.digits or {}
7168                         Babel.digits[\the\localeid] =

```

```

7169         table.pack(string.utfvalue('\bbl@cl{dgnat}'))
7170     if not Babel.numbers then
7171         function Babel.numbers(head)
7172             local LOCALE = Babel.attr_locale
7173             local GLYPH = node.id'glyph'
7174             local inmath = false
7175             for item in node.traverse(head) do
7176                 if not inmath and item.id == GLYPH then
7177                     local temp = node.get_attribute(item, LOCALE)
7178                     if Babel.digits[temp] then
7179                         local chr = item.char
7180                         if chr > 47 and chr < 58 then
7181                             item.char = Babel.digits[temp][chr-47]
7182                         end
7183                     end
7184                 elseif item.id == node.id'math' then
7185                     inmath = (item.subtype == 0)
7186                 end
7187             end
7188             return head
7189         end
7190     end
7191 }}%
7192 \fi
7193 % == transforms ==
7194 \ifx\bbl@KVP@transforms\@nnil\else
7195     \def\bbl@elt##1##2##3{%
7196         \in@{$transforms.}{##1}%
7197         \ifin@
7198             \def\bbl@tempa{##1}%
7199             \bbl@replace\bbl@tempa{transforms.}{}%
7200             \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
7201         \fi}%
7202     \bbl@exp{%
7203         \\bbl@ifblank{\bbl@cl{dgnat}}%
7204         {\let\\bbl@tempa\relax}%
7205         {\def\\bbl@tempa{%
7206             \\bbl@elt{transforms.prehyphenation}%
7207             {digits.native.1.0}{([0-9])}%
7208             \\bbl@elt{transforms.prehyphenation}%
7209             {digits.native.1.1}{string={1string|0123456789|string|\bbl@cl{dgnat}}}}}%
7210     \ifx\bbl@tempa\relax\else
7211         \toks@\expandafter\expandafter\expandafter{%
7212             \csname bbl@inidata@\language\endcsname}%
7213         \bbl@csarg\edef{inidata@\language}{%
7214             \unexpanded\expandafter{\bbl@tempa}%
7215             \the\toks@}%
7216     \fi
7217     \csname bbl@inidata@\language\endcsname
7218     \bbl@release@transforms\relax % \relax closes the last item.
7219 \fi}

```

Start tabular here:

```

7220 \def\localerestoredirs{%
7221     \ifcase\bbl@thetextdir
7222         \ifnum\textdirection=\z@\else\textdirection=\z@\fi
7223     \else
7224         \ifnum\textdirection=\@ne\else\textdirection=\@ne\fi
7225     \fi
7226     \ifcase\bbl@thepardir
7227         \ifnum\pardirection=\z@\else\pardirection=\z@\bodydirection=\z@\fi
7228     \else
7229         \ifnum\pardirection=\@ne\else\pardirection=\@ne\bodydirection=\@ne\fi

```

```

7230 \fi}
7231 %
7232 \IfBabelLayout{tabular}%
7233 {\chardef\bbl@tabular@mode\z}% All RTL
7234 {\IfBabelLayout{lrtabular}%
7235 {\chardef\bbl@tabular@mode\@ne}%
7236 {\chardef\bbl@tabular@mode\z}}% Mixed, with LTR cols
7237 %
7238 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
7239 % Redefine: vrules mess up dirs (why?).
7240 \AtBeginDocument{\def\@arstrut{\relax\copy\@arstrutbox}}%
7241 \ifcase\bbl@tabular@mode\else % 1 = Mixed
7242 \let\bbl@parabefore\relax
7243 \AddToHook{para/before}{\bbl@parabefore}
7244 \AtBeginDocument{%
7245 \bbl@replace@tabular{\hbox\bgroup}{\hbox bdir\z@\bgroup}%
7246 \ifnum\bbl@tabular@mode=\@ne
7247 \bbl@ifunset{@tabclassz}{\{ %
7248 \bbl@exp{% Hide conditionals
7249 \\\bbl@sreplace\\\@tabclassz
7250 {\<ifcase>\\\@chnum}%
7251 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
7252 \@ifpackageloaded{colortbl}%
7253 {\bbl@sreplace\@classz
7254 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
7255 {\@ifpackageloaded{array}%
7256 {\bbl@exp{% Hide conditionals
7257 \\\bbl@sreplace\\\@classz
7258 {\<ifcase>\\\@chnum}%
7259 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
7260 \\\bbl@sreplace\\\@classz
7261 {\\\do@row@strut\<fi>{\\\do@row@strut\<fi>\egroup}}}%
7262 {}}}%
7263 \fi}%
7264 \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.

```

7265 \AtBeginDocument{%
7266 \@ifpackageloaded{multicol}%
7267 {\toks\expandafter{\multi@column@out}%
7268 \edef\multi@column@out{\bodydir\pagedir\the\toks}}%
7269 {}%
7270 \@ifpackageloaded{paracol}%
7271 {\edef\pcol@output{%
7272 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7273 {}%
7274 \fi

```

Finish here if there in no layout.

```
7275 \ifx\bbl@opt@layout\@nnil\endinput\fi
```

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

```

7276 \chardef\bbl@trace@vboxdir\z@
7277 \ifnum\bbl@bidimode>\z@ % Any bidi=
7278 \IfBabelLayout{nopars}{\{
7279 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7280 \IfBabelLayout{pars}
7281 {\chardef\bbl@trace@vboxdir\@ne
7282 \def\@hangfrom#1{%
7283 \setbox\@tempboxa\hbox{\{#1}}%
7284 \hangindent\wd\@tempboxa
7285 \ifnum\bbl@vbox@bodydir=\pardirection\else

```

```

7286      \shapemode\@ne
7287      \fi
7288      \noindent\box\@tempboxa}}
7289      {}
7290 \fi

```

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

```

7291 \IfBabelLayout{lists}
7292 {\chardef\bbL@trace@vboxdir\@ne
7293  \let\bbL@OL@list\list
7294  \bbL@sreplace\list{\parshape}{\bbL@listparshape}%
7295  \let\bbL@NL@list\list
7296  \def\bbL@listparshape#1#2#3{%
7297    \parshape #1 #2 #3 %
7298    \ifnum\bbL@vbox@bodydir=\pardirection\else
7299      \shapemode\tw@
7300    \fi}}
7301 {}
7302 %
7303 \ifcase\bbL@trace@vboxdir\else
7304  \AtBeginDocument{\chardef\bbL@vbox@bodydir\pagedirection}%
7305  \def\bbL@vbox@lists{\chardef\bbL@vbox@bodydir\bodydirection}%
7306  \let\bbL@bidi@vbox\everyvbox
7307  \@nameuse{newtoks}\everyvbox % \outer in Plain
7308  \everyvbox\expandafter{\the\bbL@bidi@vbox}%
7309  \bbL@bidi@vbox{\bbL@vbox@lists\the\everyvbox}%
7310 \fi
7311 %
7312 \IfBabelLayout{graphics}
7313 {\let\bbL@pictresetdir\relax
7314  \def\bbL@pictsetdir#1{%
7315    \ifcase\bbL@thetextdir
7316      \let\bbL@pictresetdir\relax
7317    \else
7318      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7319        \or\textdir TLT
7320        \else\bodydir TLT \textdir TLT
7321      \fi
7322      % \(\text|par)dir required in pgf:
7323      \def\bbL@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7324    \fi}%
7325  \AddToHook{env/picture/begin}{\bbL@pictsetdir\tw@}%
7326  \directlua{
7327    Babel.get_picture_dir = true
7328    Babel.picture_has_bidi = 0
7329    %
7330    function Babel.picture_dir (head)
7331      if not Babel.get_picture_dir then return head end
7332      if Babel.hlist_has_bidi(head) then
7333        Babel.picture_has_bidi = 1
7334      end
7335      return head
7336    end
7337    luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7338      "Babel.picture_dir")
7339  }%
7340  \AddToHook{package/graphics/after}{%
7341    \bbL@exp{%
7342      \\bbL@sreplace\\rotatebox{\hbox{{\string#2}}}
7343      {\hbox bdir\z@{\hbox bdir\<ifcase>\bbL@thetextdir\z@\<else>\@ne\<fi>{{\string#2}}}}}}
7344  \AddToHook{package/graphicx/after}{%
7345    \bbL@exp{%

```

```

7346     \\bbl@sreplace\\Grot@box@std{\hbox{\string#2}}
7347     {\hbox bdir\z@\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>\string#2}}}%
7348     \\bbl@sreplace\\Grot@box@kv{\hbox{\string#3}}
7349     {\hbox bdir\z@\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>\string#3}}}%
7350     \\bbl@sreplace\\Grot@box{\hbox{\hbox bdir\z@}}
7351 \AtBeginDocument{%
7352   \long\def\put(#1,#2)#3{%
7353     \@killglue
7354     % Try:
7355     \ifx\bbl@pictresetdir\relax
7356       \def\bbl@tempc{0}%
7357     \else
7358       \directlua{
7359         Babel.get_picture_dir = true
7360         Babel.picture_has_bidi = 0
7361       }%
7362       \setbox\z@\hb@xt@\z@{%
7363         \@defaultunitsset\@tempdimc{#1}\unitlength
7364         \kern\@tempdimc
7365         #3\hss}%
7366       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
7367     \fi
7368     % Do:
7369     \@defaultunitsset\@tempdimc{#2}\unitlength
7370     \raise\@tempdimc\hb@xt@\z@{%
7371       \@defaultunitsset\@tempdimc{#1}\unitlength
7372       \kern\@tempdimc
7373       {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
7374     \ignorespaces}%
7375   \MakeRobust\put}%
7376 \AtBeginDocument
7377 {
7378   \AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
7379   \ifx\pgfpicture\undefined\else
7380     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
7381     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
7382     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
7383   \fi
7384   \ifx\tikzpicture\undefined\else
7385     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
7386     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
7387     \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
7388     \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
7389   \fi
7390 }
7391 {}

```

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.

```

7391 \IfBabelLayout{counters*}%
7392 {\bbl@add\bbl@opt@layout{.counters.}%
7393   \directlua{
7394     luatexbase.add_to_callback("process_output_buffer",
7395       Babel.discard_sublr , "Babel.discard_sublr") }%
7396   }{}
7397 \IfBabelLayout{counters}%
7398 {\let\bbl@latinarabic=\@arabic
7399   \let\bbl@OL@@arabic=\@arabic
7400   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7401   \ifpackagewith{babel}{bidi=default}%
7402     {\let\bbl@asciroman=\@roman
7403       \let\bbl@OL@@roman=\@roman
7404       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%

```

```

7405 \let\bbl@asciiRoman=\@Roman
7406 \let\bbl@OL@@roman\@Roman
7407 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7408 \let\bbl@OL@labelenumii\labelenumii
7409 \def\labelenumii{}\theenumii}%
7410 \let\bbl@OL@p@enumiii\p@enumiii
7411 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}

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.

7412 \IfBabelLayout{extras}%
7413 {\bbl@ncarg\let\bbl@OL@underline{underline }%
7414 \bbl@carg\bbl@sreplace{underline }\hbox}%
7415 {\def\bbl@insidemath{0}\hbox bdir\ifcase\bbl@thetextdir\z@else\@one\fi}%
7416 \let\bbl@OL@LaTeXe\LaTeXe
7417 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7418 \if b\expandafter\@car\f@series\@nil\boldmath\fi
7419 \babelsublr{\LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}}%
7420 {}
7421 (/luatex)

```

11.13 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.

```

7422 (*transforms)
7423 Babel.linebreaking.replacements = {}
7424 Babel.linebreaking.replacements[0] = {} -- pre
7425 Babel.linebreaking.replacements[1] = {} -- post
7426
7427 function Babel.tovalue(v)
7428   if type(v) == 'table' then
7429     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7430   else
7431     return v
7432   end
7433 end
7434
7435 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr_hboxed'
7436
7437 function Babel.set_hboxed(head, gc)
7438   for item in node.traverse(head) do
7439     node.set_attribute(item, Babel.attr_hboxed, 1)
7440   end
7441   return head
7442 end
7443
7444 Babel.fetch_subtext = {}
7445
7446 Babel.ignore_pre_char = function(node)
7447   return (node.lang == Babel.nohyphenation)
7448 end
7449
7450 Babel.show_transforms = false

```

```

7451
7452 -- Merging both functions doesn't seem feasible, because there are too
7453 -- many differences.
7454 Babel.fetch_subtext[0] = function(head)
7455     local word_string = ''
7456     local word_nodes = {}
7457     local lang
7458     local item = head
7459     local inmath = false
7460
7461     while item do
7462
7463         if item.id == 11 then
7464             inmath = (item.subtype == 0)
7465         end
7466
7467         if inmath then
7468             -- pass
7469
7470         elseif item.id == 29 then
7471             local locale = node.get_attribute(item, Babel.attr_locale)
7472
7473             if lang == locale or lang == nil then
7474                 lang = lang or locale
7475                 if Babel.ignore_pre_char(item) then
7476                     word_string = word_string .. Babel.us_char
7477                 else
7478                     if node.has_attribute(item, Babel.attr_hboxed) then
7479                         word_string = word_string .. Babel.us_char
7480                     else
7481                         word_string = word_string .. unicode.utf8.char(item.char)
7482                     end
7483                 end
7484                 word_nodes[#word_nodes+1] = item
7485             else
7486                 break
7487             end
7488
7489         elseif item.id == 12 and item.subtype == 13 then
7490             if node.has_attribute(item, Babel.attr_hboxed) then
7491                 word_string = word_string .. Babel.us_char
7492             else
7493                 word_string = word_string .. ' '
7494             end
7495             word_nodes[#word_nodes+1] = item
7496
7497         -- Ignore leading unrecognized nodes, too.
7498         elseif word_string ~= '' then
7499             word_string = word_string .. Babel.us_char
7500             word_nodes[#word_nodes+1] = item -- Will be ignored
7501         end
7502
7503         item = item.next
7504     end
7505
7506     -- Here and above we remove some trailing chars but not the
7507     -- corresponding nodes. But they aren't accessed.
7508     if word_string:sub(-1) == ' ' then
7509         word_string = word_string:sub(1,-2)
7510     end
7511     if Babel.show_transforms then texio.write_nl(word_string) end
7512     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7513     return word_string, word_nodes, item, lang

```

```

7514 end
7515
7516 Babel.fetch_subtext[1] = function(head)
7517     local word_string = ''
7518     local word_nodes = {}
7519     local lang
7520     local item = head
7521     local inmath = false
7522
7523     while item do
7524
7525         if item.id == 11 then
7526             inmath = (item.subtype == 0)
7527         end
7528
7529         if inmath then
7530             -- pass
7531
7532         elseif item.id == 29 then
7533             if item.lang == lang or lang == nil then
7534                 lang = lang or item.lang
7535                 if node.has_attribute(item, Babel.attr_hboxed) then
7536                     word_string = word_string .. Babel.us_char
7537                 elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7538                     word_string = word_string .. Babel.us_char
7539                 else
7540                     word_string = word_string .. unicode.utf8.char(item.char)
7541                 end
7542                 word_nodes[#word_nodes+1] = item
7543             else
7544                 break
7545             end
7546
7547         elseif item.id == 7 and item.subtype == 2 then
7548             if node.has_attribute(item, Babel.attr_hboxed) then
7549                 word_string = word_string .. Babel.us_char
7550             else
7551                 word_string = word_string .. '='
7552             end
7553             word_nodes[#word_nodes+1] = item
7554
7555         elseif item.id == 7 and item.subtype == 3 then
7556             if node.has_attribute(item, Babel.attr_hboxed) then
7557                 word_string = word_string .. Babel.us_char
7558             else
7559                 word_string = word_string .. '|'
7560             end
7561             word_nodes[#word_nodes+1] = item
7562
7563             -- (1) Go to next word if nothing was found, and (2) implicitly
7564             -- remove leading USs.
7565             elseif word_string == '' then
7566                 -- pass
7567
7568             -- This is the responsible for splitting by words.
7569             elseif (item.id == 12 and item.subtype == 13) then
7570                 break
7571
7572             else
7573                 word_string = word_string .. Babel.us_char
7574                 word_nodes[#word_nodes+1] = item -- Will be ignored
7575             end
7576

```

```

7577     item = item.next
7578 end
7579 if Babel.show_transforms then texio.write_nl(word_string) end
7580 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7581 return word_string, word_nodes, item, lang
7582 end
7583
7584 -- LuaTeX keeps its own pointer to the last node of the list passed to
7585 -- the hyphenate callback and uses it afterwards (line_break and
7586 -- filtered_hpack), so a transform freeing that node makes TeX use
7587 -- freed memory. Before transforming, the last node is replaced with a
7588 -- copy, and the original is kept out of the list, alive and with no
7589 -- links, and freed at the next call (new_ligkern walks from such a
7590 -- node to the actual end of the list).
7591
7592 do
7593     local stale_tail
7594
7595     function Babel.protect_tail(head, tail)
7596         if stale_tail then
7597             node.free(stale_tail)
7598             stale_tail = nil
7599         end
7600         if not tail or tail == head then return end
7601         local prev = tail.prev
7602         if not prev or prev.next ~= tail then
7603             prev = head
7604             while prev.next ~= tail do prev = prev.next end
7605         end
7606         local copy = node.copy(tail)
7607         prev.next = copy
7608         copy.prev = prev
7609         copy.next = nil
7610         tail.prev = nil
7611         tail.next = nil
7612         stale_tail = tail
7613     end
7614 end
7615
7616 function Babel.pre_hyphenate_replace(head)
7617     Babel.hyphenate_replace(head, 0)
7618 end
7619
7620 function Babel.post_hyphenate_replace(head)
7621     Babel.hyphenate_replace(head, 1)
7622 end
7623
7624 Babel.us_char = string.char(31)
7625
7626 function Babel.hyphenate_replace(head, mode)
7627     local u = unicode.utf8
7628     local lbkr = Babel.linebreaking.replacements[mode]
7629     local tovalue = Babel.tovalue
7630
7631     local word_head = head
7632
7633     -- Removed nodes are freed at the end of each match, because 'data'
7634     -- may still refer to them. Keyed by the direct node, so none is
7635     -- freed twice.
7636     local removed = {}
7637     local function remove_node(n)
7638         if n then
7639             node.remove(head, n)

```

```

7640     removed[node.direct.todirect(n)] = true
7641   end
7642 end
7643
7644 if Babel.show_transforms then
7645   texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7646 end
7647
7648 while true do -- for each subtext block
7649
7650   local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7651
7652   if Babel.debug then
7653     print()
7654     print((mode == 0) and '@@@<' or '@@@>', w)
7655   end
7656
7657   if nw == nil and w == '' then break end
7658
7659   if not lang then goto next end
7660   if not lbkr[lang] then goto next end
7661
7662   -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7663   -- loops are nested.
7664   for k=1, #lbkr[lang] do
7665     local p = lbkr[lang][k].pattern
7666     local r = lbkr[lang][k].replace
7667     local attr = lbkr[lang][k].attr or -1
7668
7669     if Babel.debug then
7670       print('*****', p, mode)
7671     end
7672
7673     -- This variable is set in some cases below to the first *byte*
7674     -- after the match, either as found by u.match (faster) or the
7675     -- computed position based on sc if w has changed.
7676     local last_match = 0
7677     local step = 0
7678
7679     -- For every match.
7680     while true do
7681       if Babel.debug then
7682         print('====')
7683       end
7684       local new -- used when inserting and removing nodes
7685       local dummy_node -- used by after
7686
7687       local matches = { u.match(w, p, last_match) }
7688
7689       if #matches < 2 then break end
7690
7691       -- Get and remove empty captures (with ()'s, which return a
7692       -- number with the position), and keep actual captures
7693       -- (from (...)), if any, in matches.
7694       local first = table.remove(matches, 1)
7695       local last = table.remove(matches, #matches)
7696       -- Non re-fetched substrings may contain \31, which separates
7697       -- subsubstrings.
7698       if string.find(w:sub(first, last-1), Babel.us_char) then break end
7699
7700       local save_last = last -- with A()BC()D, points to D
7701
7702       -- Fix offsets, from bytes to unicode. Explained above.

```

```

7703     first = u.len(w:sub(1, first-1)) + 1
7704     last  = u.len(w:sub(1, last-1)) -- now last points to C
7705
7706     -- This loop stores in a small table the nodes
7707     -- corresponding to the pattern. Used by 'data' to provide a
7708     -- predictable behavior with 'insert' (w_nodes is modified on
7709     -- the fly), and also access to 'remove'd nodes.
7710     local sc = first-1          -- Used below, too
7711     local data_nodes = {}
7712
7713     local enabled = true
7714     for q = 1, last-first+1 do
7715         data_nodes[q] = w_nodes[sc+q]
7716         if enabled
7717             and attr > -1
7718             and not node.has_attribute(data_nodes[q], attr)
7719         then
7720             enabled = false
7721         end
7722     end
7723
7724     -- This loop traverses the matched substring and takes the
7725     -- corresponding action stored in the replacement list.
7726     -- sc = the position in substr nodes / string
7727     -- rc = the replacement table index
7728     local rc = 0
7729
7730     ----- TODO. dummy_node?
7731     while rc < last-first+1 or dummy_node do -- for each replacement
7732         if Babel.debug then
7733             print('.....', rc + 1)
7734         end
7735         sc = sc + 1
7736         rc = rc + 1
7737
7738         if Babel.debug then
7739             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7740             local ss = ''
7741             for itt in node.traverse(head) do
7742                 if itt.id == 29 then
7743                     ss = ss .. unicode.utf8.char(itt.char)
7744                 else
7745                     ss = ss .. '{' .. itt.id .. '}'
7746                 end
7747             end
7748             print('*****', ss)
7749
7750         end
7751
7752         local crep = r[rc]
7753         local item = w_nodes[sc]
7754         local item_base = item
7755         local placeholder = Babel.us_char
7756         local d
7757
7758         if crep and crep.data then
7759             item_base = data_nodes[crep.data]
7760         end
7761
7762         if crep then
7763             step = crep.step or step
7764         end
7765

```

```

7766     if crep and crep.after then
7767         crep.insert = true
7768         if dummy_node then
7769             item = dummy_node
7770         else -- TODO. if there is a node after?
7771             d = node.copy(item_base)
7772             head, item = node.insert_after(head, item, d)
7773             dummy_node = item
7774         end
7775     end
7776
7777     if crep and not crep.after and dummy_node then
7778         remove_node(dummy_node)
7779         dummy_node = nil
7780     end
7781
7782     if not enabled then
7783         last_match = save_last
7784         goto next
7785
7786     elseif crep and next(crep) == nil then -- = {}
7787         if step == 0 then
7788             last_match = save_last    -- Optimization
7789         else
7790             last_match = utf8.offset(w, sc+step)
7791         end
7792         goto next
7793
7794     elseif crep == nil or crep.remove then
7795         remove_node(item)
7796         table.remove(w_nodes, sc)
7797         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7798         sc = sc - 1 -- Nothing has been inserted.
7799         last_match = utf8.offset(w, sc+1+step)
7800         goto next
7801
7802     elseif crep and crep.kashida then
7803         node.set_attribute(item,
7804             Babel.attr_kashida,
7805             crep.kashida)
7806         last_match = utf8.offset(w, sc+1+step)
7807         goto next
7808
7809     elseif crep and crep.string then
7810         local str = crep.string(matches)
7811         if str == '' then -- Gather with nil
7812             remove_node(item)
7813             table.remove(w_nodes, sc)
7814             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7815             sc = sc - 1 -- Nothing has been inserted.
7816         else
7817             local loop_first = true
7818             for s in string.utfvalues(str) do
7819                 d = node.copy(item_base)
7820                 d.char = s
7821                 if loop_first then
7822                     loop_first = false
7823                     head, new = node.insert_before(head, item, d)
7824                     if sc == 1 then
7825                         word_head = head
7826                     end
7827                     w_nodes[sc] = d
7828                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)

```

```

7829         else
7830             sc = sc + 1
7831             head, new = node.insert_before(head, item, d)
7832             table.insert(w_nodes, sc, new)
7833             w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7834         end
7835         if Babel.debug then
7836             print('.....', 'str')
7837             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7838         end
7839         end -- for
7840         remove_node(item)
7841     end -- if ''
7842     last_match = utf8.offset(w, sc+1+step)
7843     goto next
7844
7845 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7846     d = node.new(7, 3) -- (disc, regular)
7847     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7848     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7849     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7850     d.attr = item_base.attr
7851     if crep.pre == nil then -- TeXbook p96
7852         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7853     else
7854         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7855     end
7856     placeholder = '|'
7857     head, new = node.insert_before(head, item, d)
7858
7859 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7860     -- ERROR
7861
7862 elseif crep and crep.penalty then
7863     d = node.new(14, 0) -- (penalty, userpenalty)
7864     d.attr = item_base.attr
7865     d.penalty = tovalue(crep.penalty)
7866     head, new = node.insert_before(head, item, d)
7867
7868 elseif crep and crep.space then
7869     -- 655360 = 10 pt = 10 * 65536 sp
7870     d = node.new(12, 13) -- (glue, spaceskip)
7871     local quad = font.getfont(item_base.font).size or 655360
7872     node.setglue(d, tovalue(crep.space[1]) * quad,
7873                 tovalue(crep.space[2]) * quad,
7874                 tovalue(crep.space[3]) * quad)
7875     if mode == 0 then
7876         placeholder = ' '
7877     end
7878     head, new = node.insert_before(head, item, d)
7879
7880 elseif crep and crep.norule then
7881     -- 655360 = 10 pt = 10 * 65536 sp
7882     d = node.new(2, 3) -- (rule, empty) = \no*rule
7883     local quad = font.getfont(item_base.font).size or 655360
7884     d.width = tovalue(crep.norule[1]) * quad
7885     d.height = tovalue(crep.norule[2]) * quad
7886     d.depth = tovalue(crep.norule[3]) * quad
7887     head, new = node.insert_before(head, item, d)
7888
7889 elseif crep and crep.spacefactor then
7890     d = node.new(12, 13) -- (glue, spaceskip)
7891     local base_font = font.getfont(item_base.font)

```

```

7892         node.setglue(d,
7893             tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7894             tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7895             tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7896         if mode == 0 then
7897             placeholder = ' '
7898         end
7899         head, new = node.insert_before(head, item, d)
7900
7901     elseif mode == 0 and crep and crep.space then
7902         -- ERROR
7903
7904     elseif crep and crep.kern then
7905         d = node.new(13, 1) -- (kern, user)
7906         local quad = font.getfont(item_base.font).size or 655360
7907         d.attr = item_base.attr
7908         d.kern = tovalue(crep.kern) * quad
7909         head, new = node.insert_before(head, item, d)
7910
7911     elseif crep and crep.node then
7912         d = node.new(crep.node[1], crep.node[2])
7913         d.attr = item_base.attr
7914         head, new = node.insert_before(head, item, d)
7915
7916     end -- i.e., replacement cases
7917
7918     -- Shared by disc, space(factor), kern, node and penalty.
7919     if sc == 1 then
7920         word_head = head
7921     end
7922     if crep.insert then
7923         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7924         table.insert(w_nodes, sc, new)
7925         last = last + 1
7926     else
7927         w_nodes[sc] = d
7928         remove_node(item)
7929         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7930     end
7931
7932     last_match = utf8.offset(w, sc+1+step)
7933
7934     ::next::
7935
7936     end -- for each replacement
7937
7938     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7939     if Babel.debug then
7940         print('.....', '/')
7941         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7942     end
7943
7944     if dummy_node then
7945         remove_node(dummy_node)
7946         dummy_node = nil
7947     end
7948
7949     for n in pairs(removed) do
7950         node.direct.free(n)
7951     end
7952     removed = {}
7953
7954     end -- for match

```

```

7955
7956     end -- for patterns
7957
7958     ::next::
7959     word_head = nw
7960 end -- for substring
7961
7962 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7963 return head
7964 end
7965
7966 -- This table stores capture maps, numbered consecutively
7967 Babel.capture_maps = {}
7968
7969 function Babel.esc_hex_to_char(h)
7970   if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7971     tex.getcatcode(tonumber(h, 16)) ~= 12 then
7972     return string.format(["\Uchar"%X ], tonumber(h,16))
7973   else
7974     return unicode.utf8.char(tonumber(h, 16))
7975   end
7976 end
7977
7978 -- The following functions belong to the next macro
7979 function Babel.capture_func(key, cap)
7980   local ret = "[" .. cap:gsub('{{([0-9])}}', ")]..m[%1]..["] .. "]"
7981   local cnt
7982   local u = unicode.utf8
7983   ret = u.gsub(ret, '{(%x%x%x%x+)}', '\x01%\x04')
7984   ret, cnt = ret:gsub('{{([0-9])}([^\+]|(-))}', Babel.capture_func_map)
7985   ret = u.gsub(ret, '\x01(%x%x%x%x+)\x04', Babel.esc_hex_to_char)
7986   ret = ret:gsub("%[%]%%.", '')
7987   ret = ret:gsub("%.%[%]%", '')
7988   return key .. [=function(m) return ] .. ret .. [=end]]
7989 end
7990
7991 function Babel.capt_map(from, mapno)
7992   return Babel.capture_maps[mapno][from] or from
7993 end
7994
7995 -- Handle the {n|abc|ABC} syntax in captures
7996 function Babel.capture_func_map(capno, from, to)
7997   local u = unicode.utf8
7998   from = u.gsub(from, '\x01(%x%x%x%x+)\x04',
7999     function (n)
8000       return u.char(tonumber(n, 16))
8001     end)
8002   to = u.gsub(to, '\x01(%x%x%x%x+)\x04',
8003     function (n)
8004       return u.char(tonumber(n, 16))
8005     end)
8006   local froms = {}
8007   for s in string.utfcharacters(from) do
8008     table.insert(froms, s)
8009   end
8010   local cnt = 1
8011   table.insert(Babel.capture_maps, {})
8012   local mlen = table.getn(Babel.capture_maps)
8013   for s in string.utfcharacters(to) do
8014     Babel.capture_maps[mlen][froms[cnt]] = s
8015     cnt = cnt + 1
8016   end
8017   return "]"..Babel.capt_map(m[" .. capno .. "], " ..

```

```

8018         (mlen) .. ").." .. "["
8019 end
8020
8021 -- Create/Extend reversed sorted list of kashida weights:
8022 function Babel.capture_kashida(key, wt)
8023     wt = tonumber(wt)
8024     if Babel.kashida_wts then
8025         for p, q in ipairs(Babel.kashida_wts) do
8026             if wt == q then
8027                 break
8028             elseif wt > q then
8029                 table.insert(Babel.kashida_wts, p, wt)
8030                 break
8031             elseif table.getn(Babel.kashida_wts) == p then
8032                 table.insert(Babel.kashida_wts, wt)
8033             end
8034         end
8035     else
8036         Babel.kashida_wts = { wt }
8037     end
8038     return 'kashida = ' .. wt
8039 end
8040
8041 function Babel.capture_node(id, subtype)
8042     local sbt = 0
8043     for k, v in pairs(node.subtypes(id)) do
8044         if v == subtype then sbt = k end
8045     end
8046     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
8047 end
8048
8049 -- Experimental: applies prehyphenation transforms to a string (letters
8050 -- and spaces).
8051 function Babel.string_prehyphenation(str, locale)
8052     local n, head, last, res
8053     head = node.new(8, 0) -- dummy (hack just to start)
8054     last = head
8055     for s in string.utfvalues(str) do
8056         if s == 20 then
8057             n = node.new(12, 0)
8058         else
8059             n = node.new(29, 0)
8060             n.char = s
8061         end
8062         node.set_attribute(n, Babel.attr_locale, locale)
8063         last.next = n
8064         last = n
8065     end
8066     head = Babel.hyphenate_replace(head, 0)
8067     res = ''
8068     for n in node.traverse(head) do
8069         if n.id == 12 then
8070             res = res .. ' '
8071         elseif n.id == 29 then
8072             res = res .. unicode.utf8.char(n.char)
8073         end
8074     end
8075     node.flush_list(head)
8076     tex.print(res)
8077 end
8078 (/transforms)

```

11.14 Lua: Auto bidi with basic and basic-r

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

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

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

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

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

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

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

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

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

```
8079 (*basic-r)
8080 Babel.bidi_enabled = true
8081
8082 require('babel-data-bidi.lua')
8083
8084 local characters = Babel.characters
8085 local ranges = Babel.ranges
8086
8087 local DIR = node.id("dir")
8088
8089 local function dir_mark(head, from, to, outer)
8090   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
8091   local d = node.new(DIR)
8092   d.dir = '+' .. dir
8093   node.insert_before(head, from, d)
8094   d = node.new(DIR)
8095   d.dir = '-' .. dir
8096   node.insert_after(head, to, d)
8097 end
8098
8099 function Babel.bidi(head, ispar)
8100   local first_n, last_n      -- first and last char with nums
8101   local last_es              -- an auxiliary 'last' used with nums
8102   local first_d, last_d      -- first and last char in L/R block
8103   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):

```

8104 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
8105 local strong_lr = (strong == 'l') and 'l' or 'r'
8106 local outer = strong
8107
8108 local new_dir = false
8109 local first_dir = false
8110 local inmath = false
8111
8112 local last_lr
8113
8114 local type_n = ''
8115
8116 for item in node.traverse(head) do
8117
8118   -- three cases: glyph, dir, otherwise
8119   if item.id == node.id'glyph'
8120     or (item.id == 7 and item.subtype == 2) then
8121
8122     local itemchar
8123     if item.id == 7 and item.subtype == 2 then
8124       itemchar = item.replace.char
8125     else
8126       itemchar = item.char
8127     end
8128     local chardata = characters[itemchar]
8129     dir = chardata and chardata.d or nil
8130     if not dir then
8131       for nn, et in ipairs(ranges) do
8132         if itemchar < et[1] then
8133           break
8134         elseif itemchar <= et[2] then
8135           dir = et[3]
8136           break
8137         end
8138       end
8139     end
8140     dir = dir or 'l'
8141     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).

```

8142   if new_dir then
8143     attr_dir = 0
8144     for at in node.traverse(item.attr) do
8145       if at.number == Babel.attr_dir then
8146         attr_dir = at.value & 0x3
8147       end
8148     end
8149     if attr_dir == 1 then
8150       strong = 'r'
8151     elseif attr_dir == 2 then
8152       strong = 'al'
8153     else
8154       strong = 'l'
8155     end
8156     strong_lr = (strong == 'l') and 'l' or 'r'
8157     outer = strong_lr

```

```

8158         new_dir = false
8159     end
8160
8161     if dir == 'nsm' then dir = strong end          -- W1

```

Numbers. The dual $\langle al \rangle / \langle r \rangle$ system for R is somewhat cumbersome.

```

8162     dir_real = dir          -- We need dir_real to set strong below
8163     if dir == 'al' then dir = 'r' end -- W3

```

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

```

8164     if strong == 'al' then
8165         if dir == 'en' then dir = 'an' end          -- W2
8166         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
8167         strong_lr = 'r'                             -- W3
8168     end

```

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

```

8169     elseif item.id == node.id'dir' and not inmath then
8170         new_dir = true
8171         dir = nil
8172     elseif item.id == node.id'math' then
8173         inmath = (item.subtype == 0)
8174     else
8175         dir = nil          -- Not a char
8176     end

```

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

```

8177     if dir == 'en' or dir == 'an' or dir == 'et' then
8178         if dir ~= 'et' then
8179             type_n = dir
8180         end
8181         first_n = first_n or item
8182         last_n = last_es or item
8183         last_es = nil
8184     elseif dir == 'es' and last_n then -- W3+W6
8185         last_es = item
8186     elseif dir == 'cs' then          -- it's right - do nothing
8187     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
8188         if strong_lr == 'r' and type_n ~= '' then
8189             dir_mark(head, first_n, last_n, 'r')
8190         elseif strong_lr == 'l' and first_d and type_n == 'an' then
8191             dir_mark(head, first_n, last_n, 'r')
8192             dir_mark(head, first_d, last_d, outer)
8193             first_d, last_d = nil, nil
8194         elseif strong_lr == 'l' and type_n ~= '' then
8195             last_d = last_n
8196         end
8197         type_n = ''
8198         first_n, last_n = nil, nil
8199     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:

```

8200     if dir == 'l' or dir == 'r' then
8201         if dir ~= outer then
8202             first_d = first_d or item
8203             last_d = item

```

```

8204     elseif first_d and dir ~= strong_lr then
8205         dir_mark(head, first_d, last_d, outer)
8206         first_d, last_d = nil, nil
8207     end
8208 end

```

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

```

8209     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
8210         item.char = characters[item.char] and
8211             characters[item.char].m or item.char
8212     elseif (dir or new_dir) and last_lr ~= item then
8213         local mir = outer .. strong_lr .. (dir or outer)
8214         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
8215             for ch in node.traverse(node.next(last_lr)) do
8216                 if ch == item then break end
8217                 if ch.id == node.id'glyph' and characters[ch.char] then
8218                     ch.char = characters[ch.char].m or ch.char
8219                 end
8220             end
8221         end
8222     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).

```

8223     if dir == 'l' or dir == 'r' then
8224         last_lr = item
8225         strong = dir_real          -- Don't search back - best save now
8226         strong_lr = (strong == 'l') and 'l' or 'r'
8227     elseif new_dir then
8228         last_lr = nil
8229     end
8230 end

```

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

```

8231     if last_lr and outer == 'r' then
8232         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
8233             if characters[ch.char] then
8234                 ch.char = characters[ch.char].m or ch.char
8235             end
8236         end
8237     end
8238     if first_n then
8239         dir_mark(head, first_n, last_n, outer)
8240     end
8241     if first_d then
8242         dir_mark(head, first_d, last_d, outer)
8243     end

```

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

```

8244     return node.prev(head) or head
8245 end
8246 (/basic-r)

```

And here the Lua code for bidi=basic:

```

8247 (*basic)
8248 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
8249
8250 Babel.fontmap = Babel.fontmap or {}
8251 Babel.fontmap[0] = {}          -- l

```

```

8252 Babel.fontmap[1] = {}      -- r
8253 Babel.fontmap[2] = {}      -- al/an
8254
8255 -- To cancel mirroring. Also OML, OMS, U?
8256 Babel.symbol_fonts = Babel.symbol_fonts or {}
8257 Babel.symbol_fonts[font.id('tenln')] = true
8258 Babel.symbol_fonts[font.id('tenlnw')] = true
8259 Babel.symbol_fonts[font.id('tencirc')] = true
8260 Babel.symbol_fonts[font.id('tencircw')] = true
8261
8262 Babel.bidi_enabled = true
8263 Babel.mirroring_enabled = true
8264
8265 require('babel-data-bidi.lua')
8266
8267 local characters = Babel.characters
8268 local ranges = Babel.ranges
8269
8270 local DIR = node.id('dir')
8271 local GLYPH = node.id('glyph')
8272
8273 local function insert_implicit(head, state, outer)
8274   local new_state = state
8275   if state.sim and state.eim and state.sim ~= state.eim then
8276     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
8277     local d = node.new(DIR)
8278     d.dir = '+' .. dir
8279     node.insert_before(head, state.sim, d)
8280     local d = node.new(DIR)
8281     d.dir = '-' .. dir
8282     node.insert_after(head, state.eim, d)
8283   end
8284   new_state.sim, new_state.eim = nil, nil
8285   return head, new_state
8286 end
8287
8288 local function insert_numeric(head, state)
8289   local new
8290   local new_state = state
8291   if state.san and state.ean and state.san ~= state.ean then
8292     local d = node.new(DIR)
8293     d.dir = '+TLT'
8294     _, new = node.insert_before(head, state.san, d)
8295     if state.san == state.sim then state.sim = new end
8296     local d = node.new(DIR)
8297     d.dir = '-TLT'
8298     _, new = node.insert_after(head, state.ean, d)
8299     if state.ean == state.eim then state.eim = new end
8300   end
8301   new_state.san, new_state.ean = nil, nil
8302   return head, new_state
8303 end
8304
8305 local function glyph_not_symbol_font(node)
8306   if node.id == GLYPH then
8307     return not Babel.symbol_fonts[node.font]
8308   else
8309     return false
8310   end
8311 end
8312
8313 -- TODO - \hbox with an explicit dir can lead to wrong results
8314 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt

```

```

8315 -- was made to improve the situation, but the problem is the 3-dir
8316 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8317 -- well.
8318
8319 function Babel.bidi(head, ispar, hdir)
8320   local d    -- d is used mainly for computations in a loop
8321   local prev_d = ''
8322   local new_d = false
8323
8324   local nodes = {}
8325   local outer_first = nil
8326   local inmath = false
8327
8328   local glue_d = nil
8329   local glue_i = nil
8330
8331   local has_en = false
8332   local first_et = nil
8333
8334   local has_hyperlink = false
8335
8336   local ATDIR = Babel.attr_dir
8337   local attr_d, temp
8338   local locale_d
8339
8340   local save_outer
8341   local locale_d = node.get_attribute(head, ATDIR)
8342   if locale_d then
8343     locale_d = locale_d & 0x3
8344     save_outer = (locale_d == 0 and 'l') or
8345                  (locale_d == 1 and 'r') or
8346                  (locale_d == 2 and 'al')
8347   elseif ispar then    -- Or error? Shouldn't happen
8348     -- when the callback is called, we are just _after_ the box,
8349     -- and the textdir is that of the surrounding text
8350     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8351   else    -- Empty box
8352     save_outer = ('TRT' == hdir) and 'r' or 'l'
8353   end
8354   local outer = save_outer
8355   local last = outer
8356   -- 'al' is only taken into account in the first, current loop
8357   if save_outer == 'al' then save_outer = 'r' end
8358
8359   local fontmap = Babel.fontmap
8360
8361   for item in node.traverse(head) do
8362
8363     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8364     locale_d = node.get_attribute(item, ATDIR)
8365     node.set_attribute(item, ATDIR, 0x80)
8366
8367     -- In what follows, #node is the last (previous) node, because the
8368     -- current one is not added until we start processing the neutrals.
8369     -- three cases: glyph, dir, otherwise
8370     if glyph_not_symbol_font(item)
8371       or (item.id == 7 and item.subtype == 2) then
8372
8373       if inmath then goto nextnode end
8374
8375       if locale_d == 0x80 then goto nextnode end
8376       locale_d = locale_d or ((save_outer=='l') and 0 or 1)
8377

```

```

8378     local d_font = nil
8379     local item_r
8380     if item.id == 7 and item.subtype == 2 then
8381         item_r = item.replace -- automatic discs have just 1 glyph
8382     else
8383         item_r = item
8384     end
8385
8386     local chardata = characters[item_r.char]
8387     d = chardata and chardata.d or nil
8388     if not d or d == 'nsm' then
8389         for nn, et in ipairs(ranges) do
8390             if item_r.char < et[1] then
8391                 break
8392             elseif item_r.char <= et[2] then
8393                 if not d then d = et[3]
8394                 elseif d == 'nsm' then d_font = et[3]
8395                 end
8396                 break
8397             end
8398         end
8399     end
8400     d = d or 'l'
8401
8402     -- A short 'pause' in bidi for mapfont
8403     -- %%% TODO. move if fontmap here
8404     d_font = d_font or d
8405     d_font = (d_font == 'l' and 0) or
8406             (d_font == 'nsm' and 0) or
8407             (d_font == 'r' and 1) or
8408             (d_font == 'al' and 2) or
8409             (d_font == 'an' and 2) or nil
8410     if d_font and fontmap and fontmap[d_font][item_r.font] then
8411         item_r.font = fontmap[d_font][item_r.font]
8412     end
8413
8414     if new_d then
8415         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8416         if inmath then
8417             attr_d = 0
8418         else
8419             attr_d = locale_d & 0x3
8420         end
8421         if attr_d == 1 then
8422             outer_first = 'r'
8423             last = 'r'
8424         elseif attr_d == 2 then
8425             outer_first = 'r'
8426             last = 'al'
8427         else
8428             outer_first = 'l'
8429             last = 'l'
8430         end
8431         outer = last
8432         has_en = false
8433         first_et = nil
8434         new_d = false
8435     end
8436
8437     if glue_d then
8438         if (d == 'l' and 'l' or 'r') ~= glue_d then
8439             table.insert(nodes, {glue_i, 'on', nil})
8440         end

```

```

8441         glue_d = nil
8442         glue_i = nil
8443     end
8444
8445     elseif item.id == DIR then
8446         if inmath then goto nextnode end
8447         d = nil
8448         new_d = true
8449
8450     elseif item.id == node.id'glue' and item.subtype == 13 then
8451         if inmath then goto nextnode end
8452         glue_d = d
8453         glue_i = item
8454         d = nil
8455
8456     elseif item.id == node.id'math' then
8457         if item.subtype == 0 then -- mathon
8458             inmath = true
8459             d = 'on'
8460             table.insert(nodes, {item, 'on', outer_first})
8461             outer_first = nil
8462             goto nextnode
8463         else -- mathoff
8464             inmath = false
8465         end
8466
8467     elseif item.id == 8 and item.subtype == 19 then
8468         has_hyperlink = true
8469
8470     else
8471         d = nil
8472     end
8473
8474     -- AL <= EN/ET/ES      -- W2 + W3 + W6
8475     if last == 'al' and d == 'en' then
8476         d = 'an'          -- W3
8477     elseif last == 'al' and (d == 'et' or d == 'es') then
8478         d = 'on'          -- W6
8479     end
8480
8481     -- EN + CS/ES + EN      -- W4
8482     if d == 'en' and #nodes >= 2 then
8483         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8484             and nodes[#nodes-1][2] == 'en' then
8485             nodes[#nodes][2] = 'en'
8486         end
8487     end
8488
8489     -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
8490     if d == 'an' and #nodes >= 2 then
8491         if (nodes[#nodes][2] == 'cs')
8492             and nodes[#nodes-1][2] == 'an' then
8493             nodes[#nodes][2] = 'an'
8494         end
8495     end
8496
8497     -- ET/EN                -- W5 + W7->l / W6->on
8498     if d == 'et' then
8499         first_et = first_et or (#nodes + 1)
8500     elseif d == 'en' then
8501         has_en = true
8502         first_et = first_et or (#nodes + 1)
8503     elseif first_et then -- d may be nil here !

```

```

8504     if has_en then
8505         if last == 'l' then
8506             temp = 'l'      -- W7
8507         else
8508             temp = 'en'     -- W5
8509         end
8510     else
8511         temp = 'on'        -- W6
8512     end
8513     for e = first_et, #nodes do
8514         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8515     end
8516     first_et = nil
8517     has_en = false
8518 end
8519
8520 -- Force mathdir in math if ON (currently works as expected only
8521 -- with 'l')
8522
8523 if inmath and d == 'on' then
8524     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8525 end
8526
8527 if d then
8528     if d == 'al' then
8529         d = 'r'
8530         last = 'al'
8531     elseif d == 'l' or d == 'r' then
8532         last = d
8533     end
8534     prev_d = d
8535     table.insert(nodes, {item, d, outer_first})
8536 end
8537
8538 outer_first = nil
8539
8540 ::nextnode::
8541
8542 end -- for each node
8543
8544 -- TODO -- repeated here in case EN/ET is the last node. Find a
8545 -- better way of doing things:
8546 if first_et then      -- dir may be nil here !
8547     if has_en then
8548         if last == 'l' then
8549             temp = 'l'      -- W7
8550         else
8551             temp = 'en'     -- W5
8552         end
8553     else
8554         temp = 'on'        -- W6
8555     end
8556     for e = first_et, #nodes do
8557         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8558     end
8559 end
8560
8561 -- dummy node, to close things
8562 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8563
8564 ----- NEUTRAL -----
8565
8566 outer = save_outer

```

```

8567 last = outer
8568
8569 local first_on = nil
8570
8571 for q = 1, #nodes do
8572     local item
8573
8574     local outer_first = nodes[q][3]
8575     outer = outer_first or outer
8576     last = outer_first or last
8577
8578     local d = nodes[q][2]
8579     if d == 'an' or d == 'en' then d = 'r' end
8580     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8581
8582     if d == 'on' then
8583         first_on = first_on or q
8584     elseif first_on then
8585         if last == d then
8586             temp = d
8587         else
8588             temp = outer
8589         end
8590         for r = first_on, q - 1 do
8591             nodes[r][2] = temp
8592             item = nodes[r][1] -- MIRRORING
8593             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8594                 and temp == 'r' and characters[item.char] then
8595                 local font_mode = ''
8596                 if item.font > 0 and font.fonts[item.font].properties then
8597                     font_mode = font.fonts[item.font].properties.mode
8598                 end
8599                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8600                     item.char = characters[item.char].m or item.char
8601                 end
8602             end
8603         end
8604         first_on = nil
8605     end
8606
8607     if d == 'r' or d == 'l' then last = d end
8608 end
8609
8610 ----- IMPLICIT, REORDER -----
8611
8612 outer = save_outer
8613 last = outer
8614
8615 local state = {}
8616 state.has_r = false
8617
8618 for q = 1, #nodes do
8619
8620     local item = nodes[q][1]
8621
8622     outer = nodes[q][3] or outer
8623
8624     local d = nodes[q][2]
8625
8626     if d == 'nsm' then d = last end -- W1
8627     if d == 'en' then d = 'an' end
8628     local isdir = (d == 'r' or d == 'l')
8629

```

```

8630   if outer == 'l' and d == 'an' then
8631       state.san = state.san or item
8632       state.ean = item
8633   elseif state.san then
8634       head, state = insert_numeric(head, state)
8635   end
8636
8637   if outer == 'l' then
8638       if d == 'an' or d == 'r' then      -- im -> implicit
8639           if d == 'r' then state.has_r = true end
8640           state.sim = state.sim or item
8641           state.eim = item
8642       elseif d == 'l' and state.sim and state.has_r then
8643           head, state = insert_implicit(head, state, outer)
8644       elseif d == 'l' then
8645           state.sim, state.eim, state.has_r = nil, nil, false
8646       end
8647   else
8648       if d == 'an' or d == 'l' then
8649           if nodes[q][3] then -- nil except after an explicit dir
8650               state.sim = item -- so we move sim 'inside' the group
8651           else
8652               state.sim = state.sim or item
8653           end
8654           state.eim = item
8655       elseif d == 'r' and state.sim then
8656           head, state = insert_implicit(head, state, outer)
8657       elseif d == 'r' then
8658           state.sim, state.eim = nil, nil
8659       end
8660   end
8661
8662   if isdir then
8663       last = d      -- Don't search back - best save now
8664   elseif d == 'on' and state.san then
8665       state.san = state.san or item
8666       state.ean = item
8667   end
8668
8669 end
8670
8671 head = node.prev(head) or head
8672 % \end{macrocode}
8673 %
8674 % Now direction nodes has been distributed with relation to characters
8675 % and spaces, we need to take into account \TeX-specific elements in
8676 % the node list, to move them at an appropriate place. Firstly, with
8677 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8678 % that the latter are still discardable.
8679 %
8680 % \begin{macrocode}
8681 --- FIXES ---
8682 if has_hyperlink then
8683     local flag, linking = 0, 0
8684     for item in node.traverse(head) do
8685         if item.id == DIR then
8686             if item.dir == '+TRT' or item.dir == '+TLT' then
8687                 flag = flag + 1
8688             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8689                 flag = flag - 1
8690             end
8691         elseif item.id == 8 and item.subtype == 19 then
8692             linking = flag

```

```

8693     elseif item.id == 8 and item.subtype == 20 then
8694         if linking > 0 then
8695             if item.prev.id == DIR and
8696                 (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8697                 local old = item.prev
8698                 d = node.new(DIR)
8699                 d.dir = old.dir
8700                 head = node.remove(head, old)
8701                 node.free(old)
8702                 node.insert_after(head, item, d)
8703             end
8704         end
8705         linking = 0
8706     end
8707 end
8708 end
8709
8710 local item, next_item = head
8711 while item do
8712     next_item = item.next
8713     if item.id == 10 then
8714         local p = item
8715         while p.prev and p.prev.id == 14 do
8716             p = p.prev
8717         end
8718         if p ~= item then
8719             head = node.remove(head, item)
8720             head = node.insert_before(head, p, item)
8721         end
8722     end
8723     item = next_item
8724 end
8725
8726 return head
8727 end
8728
8728 function Babel.unset_atdir(head)
8729     local ATDIR = Babel.attr_dir
8730     for item in node.traverse(head) do
8731         node.set_attribute(item, ATDIR, 0x80)
8732     end
8733     return head
8734 end

```

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

```

8735 local GLUE      = node.id'glue'
8736 local PENALTY   = node.id'penalty'
8737 local HLIST     = node.id'hlist'
8738 local LINE      = table.swapped(node.subtypes("hlist")).line
8739
8740 local characters = Babel.characters
8741 local ranges     = Babel.ranges
8742
8743 -- L1's whitespace set: WS + isolate formats + (per the UBA reference) the
8744 -- explicit embedding/override formats + BN.
8745 -- Babel does not use embedding/override/isolates. In fact, their use
8746 -- is discouraged by the UAX#9. To be removed.
8747 local ll_whitespace = {
8748     ws = true, bn = true,
8749     lre = true, rle = true, lro = true, rlo = true, pdf = true,
8750     lri = true, rli = true, fsi = true, pdi = true,
8751 }
8752

```

```

8753 local function bidi_class(c)
8754   local cd = characters[c]
8755   local d = cd and cd.d
8756   if not d then
8757     for _, et in ipairs(ranges) do
8758       if c < et[1] then
8759         break
8760       elseif c <= et[2] then
8761         d = et[3]
8762         break
8763       end
8764     end
8765   end
8766   return d or 'l'
8767 end
8768
8769 -- Nodes we step over as part of a trailing-whitespace run: glue,
8770 -- penalties, and a whitespace glyph.
8771 local function skippable(n)
8772   local id = n.id
8773   if id == GLUE or id == PENALTY then
8774     return true
8775   elseif id == GLYPH then
8776     return ll_whitespace[bidi_class(n.char)] == true
8777   end
8778   return false
8779 end
8780
8781 local function is_enddir(n)
8782   return n.id == DIR and n.subtype == 1
8783 end
8784
8785 -- Rule L1, clause 4, for one line: pull its trailing whitespace out past the
8786 -- closing dir run so it renders in the box (base) direction.
8787 local function reset_line(head)
8788   local last = node.slide(head) -- tail, fixes prev links
8789   -- 1. skip the tail that follows the closing dir run (rightskip/parfillskip/...)
8790   local q = last
8791   while q and skippable(q) do q = q.prev end
8792   if not (q and is_enddir(q)) then return head end
8793   local d_last = q
8794   -- 2. step back over the whole closing dir run
8795   while q and q.id == DIR do q = q.prev end
8796   -- 3. what precedes the dirs must be the trailing whitespace run
8797   if not (q and skippable(q)) then return head end
8798   local w_last = q
8799   while q and skippable(q) do q = q.prev end
8800   local before_w = q
8801   if not before_w then return head end -- whole line is whitespace; leave it
8802   local w_first = before_w.next
8803   local d_first = w_last.next
8804   local after_d = d_last.next
8805   -- detach W = [w_first .. w_last]
8806   before_w.next = d_first
8807   if d_first then d_first.prev = before_w end
8808   -- reinsert W after the dir run (outside every dir => box/base direction)
8809   d_last.next = w_first
8810   w_first.prev = d_last
8811   w_last.next = after_d
8812   if after_d then after_d.prev = w_last end
8813   return head
8814 end
8815

```

```

8816 function Babel.reset_lines(head)
8817   for line, subtype in node.traverse_id(HLIST, head) do
8818     if subtype == LINE then
8819       line.list = reset_line(line.list)
8820     end
8821   end
8822   return head
8823 end
8824
8825 luatexbase.add_to_callback('post_linebreak_filter',
8826   function(head, gc)
8827     if Babel.bidi_enabled then head = Babel.reset_lines(head) end
8828     return head
8829   end,
8830   'Babel.reset_lines')
8831 (/basic)

```

12. 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.

13. 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.

```

8832 (*nil)
8833 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8834 \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.

```

8835 \ifx\l@nil\undefined
8836   \newlanguage\l@nil
8837   \namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
8838   \let\bbl@elt\relax
8839   \edef\bbl@languages{% Add it to the list of languages
8840     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}
8841 \fi

```

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

```

8842 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

`\captionnil`

\datenil

```
8843 \let\captionsnil\@empty
8844 \let\datenil\@empty
```

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

```
8845 \def\bbl@inidata@nil{%
8846   \bbl@elt{identification}{tag.ini}{und}%
8847   \bbl@elt{identification}{load.level}{0}%
8848   \bbl@elt{identification}{charset}{utf8}%
8849   \bbl@elt{identification}{version}{1.0}%
8850   \bbl@elt{identification}{date}{2022-05-16}%
8851   \bbl@elt{identification}{name.local}{nil}%
8852   \bbl@elt{identification}{name.english}{nil}%
8853   \bbl@elt{identification}{name.babel}{nil}%
8854   \bbl@elt{identification}{tag.bcp47}{und}%
8855   \bbl@elt{identification}{language.tag.bcp47}{und}%
8856   \bbl@elt{identification}{tag.opentype}{dflt}%
8857   \bbl@elt{identification}{script.name}{Latin}%
8858   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8859   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8860   \bbl@elt{identification}{level}{1}%
8861   \bbl@elt{identification}{encodings}{}%
8862   \bbl@elt{identification}{derivate}{no}}
8863 \@namedef{bbl@tbcp@nil}{und}
8864 \@namedef{bbl@lbcp@nil}{und}
8865 \@namedef{bbl@casing@nil}{und}
8866 \@namedef{bbl@lotf@nil}{dflt}
8867 \@namedef{bbl@elname@nil}{nil}
8868 \@namedef{bbl@lname@nil}{nil}
8869 \@namedef{bbl@esname@nil}{Latin}
8870 \@namedef{bbl@sname@nil}{Latin}
8871 \@namedef{bbl@sbc@nil}{Latn}
8872 \@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.

```
8873 \ldf@finish{nil}
8874 (/nil)
```

14. 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.

```
8875 ((*Compute Julian day)) ≡
8876 \def\bbl@fpmmod#1#2{(#1-#2*floor((#1)/(#2)))}
8877 \def\bbl@cs@gregleap#1{%
8878   (\bbl@fpmmod{#1}{4} == 0) &&
8879   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8880 \def\bbl@cs@jd#1#2#3{% year, month, day
8881   \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8882     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8883     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8884     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }
8885 (/Compute Julian day)
```

14.1. Islamic

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

```
8886 (*ca-islamic)
8887 <@Compute Julian day@>
```

```

8888 % == islamic (default)
8889 % Not yet implemented
8890 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar.

```

8891 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8892 ((#3 + ceil(29.5 * (#2 - 1)) +
8893 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8894 1948439.5) - 1) }
8895 \@namedef\bbl@ca@islamic-civil++{\bbl@ca@islamicvl@x{+2}}
8896 \@namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl@x{+1}}
8897 \@namedef\bbl@ca@islamic-civil{\bbl@ca@islamicvl@x{}}
8898 \@namedef\bbl@ca@islamic-civil-{\bbl@ca@islamicvl@x{-1}}
8899 \@namedef\bbl@ca@islamic-civil--{\bbl@ca@islamicvl@x{-2}}
8900 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8901 \edef\bbl@tempa{%
8902 \fpeval{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8903 \edef#5{%
8904 \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8905 \edef#6{\fpeval{
8906 min(12,ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8907 \edef#7{\fpeval{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}

```

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

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

```

8908 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8909 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8910 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8911 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8912 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8913 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8914 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8915 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8916 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8917 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8918 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8919 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8920 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8921 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8922 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8923 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8924 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8925 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8926 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8927 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8928 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8929 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8930 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8931 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8932 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8933 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8934 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8935 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8936 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8937 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8938 65401,65431,65460,65490,65520}
8939 \@namedef\bbl@ca@islamic-umalqura+{\bbl@ca@islamcuqr@x{+1}}
8940 \@namedef\bbl@ca@islamic-umalqura{\bbl@ca@islamcuqr@x{}}
8941 \@namedef\bbl@ca@islamic-umalqura-{\bbl@ca@islamcuqr@x{-1}}
8942 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8943 \ifnum#2>2014 \ifnum#2<2038
8944 \bbl@afterfi\expandafter\@gobble

```

```

8945 \fi\fi
8946 {\bbl@error{year-out-range}{2014-2038}{}}}%
8947 \edef\bbl@tempd{\fpeval{ % (Julian) day
8948 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8949 \count@\@ne
8950 \bbl@foreach\bbl@cs@umalqura@data{%
8951 \advance\count@\@ne
8952 \ifnum##1>\bbl@tempd\else
8953 \edef\bbl@tempe{\the\count@}%
8954 \edef\bbl@tempb{##1}%
8955 \fi}%
8956 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8957 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1 ) / 12) }}% annus
8958 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8959 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8960 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8961 \bbl@add\bbl@precalendar{%
8962 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8963 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8964 \bbl@replace\bbl@ld@calendar{+}{}}%
8965 \bbl@replace\bbl@ld@calendar{-}{}}%
8966 (/ca-islamic)

```

14.2. Hebrew

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

```

8967 (*ca-hebrew)
8968 \newcount\bbl@cntcommon
8969 \def\bbl@remainder#1#2#3{%
8970 #3=#1\relax
8971 \divide #3 by #2\relax
8972 \multiply #3 by -#2\relax
8973 \advance #3 by #1\relax}%
8974 \newif\ifbbl@divisible
8975 \def\bbl@checkifdivisible#1#2{%
8976 {\countdef\tmp=0
8977 \bbl@remainder{#1}{#2}{\tmp}%
8978 \ifnum \tmp=0
8979 \global\bbl@divisibletrue
8980 \else
8981 \global\bbl@divisiblefalse
8982 \fi}}
8983 \newif\ifbbl@gregleap
8984 \def\bbl@ifgregleap#1{%
8985 \bbl@checkifdivisible{#1}{4}%
8986 \ifbbl@divisible
8987 \bbl@checkifdivisible{#1}{100}%
8988 \ifbbl@divisible
8989 \bbl@checkifdivisible{#1}{400}%
8990 \ifbbl@divisible
8991 \bbl@gregleaptrue
8992 \else
8993 \bbl@gregleapfalse
8994 \fi
8995 \else
8996 \bbl@gregleaptrue
8997 \fi
8998 \else
8999 \bbl@gregleapfalse
9000 \fi
9001 \ifbbl@gregleap}

```

```

9002 \def\bbl@gregdayspriormonths#1#2#3{%
9003     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
9004         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
9005     \bbl@ifgregleap{#2}%
9006     \ifnum #1 > 2
9007         \advance #3 by 1
9008     \fi
9009 \fi
9010 \global\bbl@cntcommon=#3}%
9011 #3=\bbl@cntcommon}
9012 \def\bbl@gregdaysprioryears#1#2{%
9013     {\countdef\tmpc=4
9014     \countdef\tmpb=2
9015     \tmpb=#1\relax
9016     \advance \tmpb by -1
9017     \tmpc=\tmpb
9018     \multiply \tmpc by 365
9019     #2=\tmpc
9020     \tmpc=\tmpb
9021     \divide \tmpc by 4
9022     \advance #2 by \tmpc
9023     \tmpc=\tmpb
9024     \divide \tmpc by 100
9025     \advance #2 by -\tmpc
9026     \tmpc=\tmpb
9027     \divide \tmpc by 400
9028     \advance #2 by \tmpc
9029     \global\bbl@cntcommon=#2\relax}%
9030 #2=\bbl@cntcommon}
9031 \def\bbl@absfromgreg#1#2#3#4{%
9032     {\countdef\tmpd=0
9033     #4=#1\relax
9034     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
9035     \advance #4 by \tmpd
9036     \bbl@gregdaysprioryears{#3}{\tmpd}%
9037     \advance #4 by \tmpd
9038     \global\bbl@cntcommon=#4\relax}%
9039 #4=\bbl@cntcommon}
9040 \newif\ifbbl@hebrleap
9041 \def\bbl@checkleaphebryear#1{%
9042     {\countdef\tmpa=0
9043     \countdef\tmpb=1
9044     \tmpa=#1\relax
9045     \multiply \tmpa by 7
9046     \advance \tmpa by 1
9047     \bbl@remainder{\tmpa}{19}{\tmpb}%
9048     \ifnum \tmpb < 7
9049         \global\bbl@hebrleaptrue
9050     \else
9051         \global\bbl@hebrleapfalse
9052     \fi}}
9053 \def\bbl@hebrlapsedmonths#1#2{%
9054     {\countdef\tmpa=0
9055     \countdef\tmpb=1
9056     \countdef\tmpc=2
9057     \tmpa=#1\relax
9058     \advance \tmpa by -1
9059     #2=\tmpa
9060     \divide #2 by 19
9061     \multiply #2 by 235
9062     \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
9063     \tmpc=\tmpb
9064     \multiply \tmpb by 12

```

```

9065 \advance #2 by \tmpb
9066 \multiply \tmpc by 7
9067 \advance \tmpc by 1
9068 \divide \tmpc by 19
9069 \advance #2 by \tmpc
9070 \global\bbl@cntcommon=#2}%
9071 #2=\bbl@cntcommon}
9072 \def\bbl@hebreleapseddays#1#2{%
9073 {\countdef\tmpa=0
9074 \countdef\tmpb=1
9075 \countdef\tmpc=2
9076 \bbl@hebreleapsedmonths{#1}{#2}%
9077 \tmpa=#2\relax
9078 \multiply \tmpa by 13753
9079 \advance \tmpa by 5604
9080 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
9081 \divide \tmpa by 25920
9082 \multiply #2 by 29
9083 \advance #2 by 1
9084 \advance #2 by \tmpa
9085 \bbl@remainder{#2}{7}{\tmpa}%
9086 \ifnum \tmpc < 19440
9087 \ifnum \tmpc < 9924
9088 \else
9089 \ifnum \tmpa=2
9090 \bbl@checkleaphebrewyear{#1}% of a common year
9091 \ifbbl@hebrleap
9092 \else
9093 \advance #2 by 1
9094 \fi
9095 \fi
9096 \fi
9097 \ifnum \tmpc < 16789
9098 \else
9099 \ifnum \tmpa=1
9100 \advance #1 by -1
9101 \bbl@checkleaphebrewyear{#1}% at the end of leap year
9102 \ifbbl@hebrleap
9103 \advance #2 by 1
9104 \fi
9105 \fi
9106 \fi
9107 \else
9108 \advance #2 by 1
9109 \fi
9110 \bbl@remainder{#2}{7}{\tmpa}%
9111 \ifnum \tmpa=0
9112 \advance #2 by 1
9113 \else
9114 \ifnum \tmpa=3
9115 \advance #2 by 1
9116 \else
9117 \ifnum \tmpa=5
9118 \advance #2 by 1
9119 \fi
9120 \fi
9121 \fi
9122 \global\bbl@cntcommon=#2\relax}%
9123 #2=\bbl@cntcommon}
9124 \def\bbl@daysinhebrewyear#1#2{%
9125 {\countdef\tmpe=12
9126 \bbl@hebreleapseddays{#1}{\tmpe}%
9127 \advance #1 by 1

```

```

9128 \bbl@hebreleaseddays{#1}{#2}%
9129 \advance #2 by -\tmpe
9130 \global\bbl@cntcommon=#2}%
9131 #2=\bbl@cntcommon}
9132 \def\bbl@hebrdayspriormonths#1#2#3{%
9133 {\countdef\tmpf= 14
9134 #3=\ifcase #1
9135      0 \or
9136      0 \or
9137      30 \or
9138      59 \or
9139      89 \or
9140      118 \or
9141      148 \or
9142      148 \or
9143      177 \or
9144      207 \or
9145      236 \or
9146      266 \or
9147      295 \or
9148      325 \or
9149      400
9150 \fi
9151 \bbl@checkleaphebyear{#2}%
9152 \ifbbl@hebrleap
9153   \ifnum #1 > 6
9154     \advance #3 by 30
9155   \fi
9156 \fi
9157 \bbl@daysinhebyear{#2}{\tmpf}%
9158 \ifnum #1 > 3
9159   \ifnum \tmpf=353
9160     \advance #3 by -1
9161   \fi
9162   \ifnum \tmpf=383
9163     \advance #3 by -1
9164   \fi
9165 \fi
9166 \ifnum #1 > 2
9167   \ifnum \tmpf=355
9168     \advance #3 by 1
9169   \fi
9170   \ifnum \tmpf=385
9171     \advance #3 by 1
9172   \fi
9173 \fi
9174 \global\bbl@cntcommon=#3\relax}%
9175 #3=\bbl@cntcommon}
9176 \def\bbl@absfromhebr#1#2#3#4{%
9177 {#4=#1\relax
9178 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
9179 \advance #4 by #1\relax
9180 \bbl@hebreleaseddays{#3}{#1}%
9181 \advance #4 by #1\relax
9182 \advance #4 by -1373429
9183 \global\bbl@cntcommon=#4\relax}%
9184 #4=\bbl@cntcommon}
9185 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
9186 {\countdef\tmpx= 17
9187 \countdef\tmpy= 18
9188 \countdef\tmpz= 19
9189 #6=#3\relax
9190 \global\advance #6 by 3761

```

```

9191 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
9192 \tmpz=1 \tmpy=1
9193 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
9194 \ifnum \tmpx > #4\relax
9195 \global\advance #6 by -1
9196 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
9197 \fi
9198 \advance #4 by -\tmpx
9199 \advance #4 by 1
9200 #5=#4\relax
9201 \divide #5 by 30
9202 \loop
9203 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
9204 \ifnum \tmpx < #4\relax
9205 \advance #5 by 1
9206 \tmpy=\tmpx
9207 \repeat
9208 \global\advance #5 by -1
9209 \global\advance #4 by -\tmpy}}
9210 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
9211 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
9212 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
9213 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
9214 \bbl@hebrfromgreg
9215 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
9216 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
9217 \edef#4{\the\bbl@hebryear}%
9218 \edef#5{\the\bbl@hebrmonth}%
9219 \edef#6{\the\bbl@hebrday}}
9220 (/ca-hebrew)

```

14.3. Persian

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

```

9221 (*ca-persian)
9222 <@Compute Julian day@>
9223 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
9224 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
9225 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
9226 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
9227 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
9228 \bbl@afterfi\expandafter\@gobble
9229 \fi\fi
9230 {\bbl@error{year-out-range}{2013-2050}{}}}%
9231 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
9232 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
9233 \edef\bbl@tempc{\fpeval{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
9234 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
9235 \ifnum\bbl@tempc<\bbl@tempb
9236 \edef\bbl@tempa{\fpeval{\bbl@tempa-1}}% go back 1 year and redo
9237 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
9238 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
9239 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
9240 \fi
9241 \edef#4{\fpeval{\bbl@tempa-621}}% set Jalali year
9242 \edef#6{\fpeval{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
9243 \edef#5{\fpeval{% set Jalali month
9244 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
9245 \edef#6{\fpeval{% set Jalali day

```

```

9246      (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))))}}
9247 (/ca-persian)

```

14.4. 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.

```

9248 (*ca-coptic)
9249 <@Compute Julian day@>
9250 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
9251   \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
9252   \edef\bbl@tempc{\fpeval{\bbl@tempd - 1825029.5}}%
9253   \edef#4{\fpeval{%
9254     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
9255   \edef\bbl@tempc{\fpeval{%
9256     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
9257   \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
9258   \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
9259 (/ca-coptic)
9260 (*ca-ethiopic)
9261 <@Compute Julian day@>
9262 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
9263   \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
9264   \edef\bbl@tempc{\fpeval{\bbl@tempd - 1724220.5}}%
9265   \edef#4{\fpeval{%
9266     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
9267   \edef\bbl@tempc{\fpeval{%
9268     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
9269   \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
9270   \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
9271 (/ca-ethiopic)

```

14.5. Julian

Based on [ReinDersh].

```

9272 (*ca-julian)
9273 <@Compute Julian day@>
9274 \def\bbl@ca@julian#1-#2-#3\@#4#5#6{%
9275   \edef\bbl@tempj{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + .5}}%
9276   \edef\bbl@tempa{\fpeval{\bbl@tempj + 32082.5}}%
9277   \edef\bbl@tempb{\fpeval{floor((4 * \bbl@tempa + 3) / 1461)}}%
9278   \edef\bbl@tempc{\fpeval{\bbl@tempa - floor(1461*\bbl@tempb/4)}}%
9279   \edef\bbl@tempd{\fpeval{floor((5 * \bbl@tempc + 2) / 153)}}%
9280   \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
9281   \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
9282   \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}%
9283 (/ca-julian)

```

The Indian National Calendar, Saka era.

```

9284 (*ca-indian)
9285 <@Compute Julian day@>
9286 \def\bbl@sakachaitra#1{%
9287   \bbl@cs@jd{#1}{3}{%
9288     \fpeval{ ((\bbl@fpmmod{#1}{4})==0 && \bbl@fpmmod{#1}{100})!=0)
9289     || \bbl@fpmmod{#1}{400})==0) ? 21 : 22 }}%
9290 \def\bbl@ca@indian#1-#2-#3\@#4#5#6{%
9291   \edef\bbl@tempa{\fpeval{\bbl@cs@jd{#1}{#2}{#3}}}%
9292   \edef\bbl@tempb{\fpeval{\bbl@sakachaitra{#1}}}%
9293   \edef\bbl@tempc{\fpeval{\bbl@tempa < \bbl@tempb ? #1-1 : #1}}%
9294   \edef\bbl@tempd{\fpeval{\bbl@tempa-\bbl@sakachaitra{\bbl@tempc}}}%
9295   \edef\bbl@tempe{\fpeval{ ((\bbl@fpmmod{\bbl@tempc}{4})==0 &&
9296     \bbl@fpmmod{\bbl@tempc}{100})!=0) || \bbl@fpmmod{\bbl@tempc}{400})==0) ? 31 : 30 }}%

```

```

9297 \edef#4{\fpeval{\bbl@tempc-78}}%
9298 \edef#5{\fpeval{\bbl@tempd<\bbl@tempe ? 1 :
9299 (\bbl@tempd-\bbl@tempe < 155 ?
9300 2+trunc((\bbl@tempd-\bbl@tempe)/31,0) :
9301 7+trunc((\bbl@tempd-\bbl@tempe-155)/30,0)}}}%
9302 \edef#6{\fpeval{\bbl@tempd<\bbl@tempe ? \bbl@tempd+1 :
9303 (\bbl@tempd-\bbl@tempe < 155 ?
9304 \bbl@fpmmod{\bbl@tempd-\bbl@tempe}{31}+1 :
9305 \bbl@fpmmod{\bbl@tempd-\bbl@tempe-155}{30}+1)}}}
9306 (/ca-indian)
9307 %

```

14.6. Buddhist

That's very simple.

```

9308 (*ca-buddhist)
9309 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
9310 \edef#4{\number\numexpr#1+543\relax}%
9311 \edef#5{#2}%
9312 \edef#6{#3}}
9313 (/ca-buddhist)
9314 %
9315 % \subsection{Chinese}
9316 %
9317 % Brute force, with the Julian day of first day of each month. The
9318 % table has been computed with the help of \textsf{python-lunardate} by
9319 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
9320 % is 2015-2044.
9321 %
9322 % \begin{macrocode}
9323 (*ca-chinese)
9324 \ExplSyntaxOn
9325 <@Compute Julian day@>
9326 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
9327 \edef\bbl@tempd{\fpeval{%
9328 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
9329 \count@ \z@
9330 \@tempcnta=2015
9331 \bbl@foreach\bbl@cs@chinese@data{%
9332 \ifnum##1>\bbl@tempd\else
9333 \advance\count@\@ne
9334 \ifnum\count@>12
9335 \count@\@ne
9336 \advance\@tempcnta\@ne\fi
9337 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
9338 \ifin@
9339 \advance\count@\m@ne
9340 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
9341 \else
9342 \edef\bbl@tempe{\the\count@}%
9343 \fi
9344 \edef\bbl@tempb{##1}%
9345 \fi}%
9346 \edef#4{\the\@tempcnta}%
9347 \edef#5{\bbl@tempe}%
9348 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
9349 \def\bbl@cs@chinese@leap{%
9350 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
9351 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
9352 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
9353 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
9354 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
9355 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%

```

```

9356 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
9357 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
9358 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
9359 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
9360 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
9361 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
9362 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
9363 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
9364 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
9365 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
9366 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
9367 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
9368 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
9369 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
9370 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
9371 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
9372 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
9373 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
9374 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
9375 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
9376 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
9377 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
9378 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
9379 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
9380 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
9381 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
9382 10896,10926,10956,10986,11015,11045,11074,11103}
9383 \ExplSyntaxOff
9384 (/ca-chinese)

```

15. Support for Plain T_EX (plain.def)

15.1. Not renaming hyphen.tex

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

```

9385 (*bplain|lplain)
9386 \catcode`\{=1 % left brace is begin-group character
9387 \catcode`\}=2 % right brace is end-group character
9388 \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).

```

9389 \openin 0 hyphen.cfg
9390 \ifeof0
9391 \else
9392 \let\input

```

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

```

9393 \def\input #1 {%
9394   \let\input\@
9395   \a hyphen.cfg
9396   \let\@undefined
9397 }
9398 \fi
9399 (/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`.

```

9400 (bplain)\a plain.tex
9401 (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.

```

9402 (bplain)\def\fmtname{babel-plain}
9403 (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.

15.2. Emulating some $\text{\LaTeX}$ features

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

```

9404 (<*Emulate LaTeX>) ≡
9405 \def\@empty{}
9406 \def\loadlocalcfg#1{%
9407   \openin0#1.cfg
9408   \ifeof0
9409     \closein0
9410   \else
9411     \closein0
9412     {\immediate\write16{*****}%
9413      \immediate\write16{* Local config file #1.cfg used}%
9414      \immediate\write16{*}%
9415     }
9416     \input #1.cfg\relax
9417   \fi
9418   \@endofldf}

```

15.3. General tools

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

```

9419 \long\def\@firstofone#1{#1}
9420 \long\def\@firstoftwo#1#2{#1}
9421 \long\def\@secondoftwo#1#2{#2}
9422 \def\@nnil{\@nil}
9423 \def\@gobbletwo#1#2{}
9424 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
9425 \def\@star@or@long#1{%
9426   \@ifstar
9427   {\let\l@ngrel@x\relax#1}%
9428   {\let\l@ngrel@x\long#1}}
9429 \let\l@ngrel@x\relax
9430 \def\@car#1#2\@nil{#1}
9431 \def\@cdr#1#2\@nil{#2}
9432 \let\@typeset@protect\relax
9433 \let\protected@edef\edef
9434 \long\def\@gobble#1{}

```

```

9435 \edef\@backslashchar{\expandafter\@gobble\string\}
9436 \def\strip@prefix#1>{}
9437 \def@gaddto@macro#1#2{%
9438     \toks@\expandafter{#1#2}%
9439     \xdef#1{\the\toks@}}
9440 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
9441 \def\@nameuse#1{\csname #1\endcsname}
9442 \def\@ifundefined#1{%
9443     \expandafter\ifx\csname#1\endcsname\relax
9444     \expandafter\@firstoftwo
9445     \else
9446     \expandafter\@secondoftwo
9447     \fi}
9448 \def\@expandtwoargs#1#2#3{%
9449     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9450 \def\zap@space#1 #2{%
9451     #1%
9452     \ifx#2\@empty\else\expandafter\zap@space\fi
9453     #2}
9454 \let\bbl@trace\@gobble
9455 \def\bbl@error#1{% Implicit #2#3#4
9456     \begingroup
9457     \catcode`\=0 \catcode`\==12 \catcode`\`=12
9458     \catcode`\^M=5 \catcode`\%=14
9459     \input errbabel.def
9460     \endgroup
9461     \bbl@error{#1}}
9462 \def\bbl@warning#1{%
9463     \begingroup
9464     \newlinechar=`^^J
9465     \def\{^^J(babel) }%
9466     \message{\#1}%
9467     \endgroup}
9468 \let\bbl@infowarn\bbl@warning
9469 \def\bbl@info#1{%
9470     \begingroup
9471     \newlinechar=`^^J
9472     \def\{^^J}%
9473     \wlog{#1}%
9474     \endgroup}

```

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

```

9475 \ifx\@preamblecmds\undefined
9476     \def\@preamblecmds{}
9477 \fi
9478 \def\@onlypreamble#1{%
9479     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9480         \@preamblecmds\do#1}}
9481 \@onlypreamble\@onlypreamble

```

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

```

9482 \def\begindocument{%
9483     \@begindocumenthook
9484     \global\let\@begindocumenthook\undefined
9485     \def\do##1{\global\let##1\@undefined}%
9486     \@preamblecmds
9487     \global\let\do\noexpand}
9488 \ifx\@begindocumenthook\undefined
9489     \def\@begindocumenthook{}
9490 \fi
9491 \@onlypreamble\@begindocumenthook
9492 \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`.

```

9493 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
9494 \@onlypreamble\AtEndOfPackage
9495 \def\@endofldf{}
9496 \@onlypreamble\@endofldf
9497 \let\bbl@afterlang\@empty
9498 \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.

```

9499 \catcode`\&=\z@
9500 \ifx&\if@files\@undefined
9501   \expandafter\let\csname if@files\expandafter\endcsname
9502     \csname iffalse\endcsname
9503 \fi
9504 \catcode`\&=4

```

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

```

9505 \def\newcommand{\@star@or@long\new@command}
9506 \def\new@command#1{%
9507   \@testopt{\@newcommand#1}0}
9508 \def\@newcommand#1[#2]{%
9509   \@ifnextchar [{\@xargdef#1[#2]}%
9510                 {\@argdef#1[#2]}}
9511 \long\def\@argdef#1[#2]#3{%
9512   \@yargdef#1\@ne{#2}{#3}}
9513 \long\def\@xargdef#1[#2][#3]#4{%
9514   \expandafter\def\expandafter#1\expandafter{%
9515     \expandafter\@protected@testopt\expandafter #1%
9516     \csname\string#1\expandafter\endcsname{#3}}%
9517   \expandafter\@yargdef \csname\string#1\endcsname
9518   \tw@{#2}{#4}}
9519 \long\def\@yargdef#1#2#3{%
9520   \@tempcnta#3\relax
9521   \advance \@tempcnta \@ne
9522   \let\@hash\relax
9523   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9524   \@tempcntb #2%
9525   \@whilenum \@tempcntb < \@tempcnta
9526   \do{%
9527     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
9528     \advance\@tempcntb \@ne}%
9529   \let\@hash###%
9530   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9531 \def\providecommand{\@star@or@long\provide@command}
9532 \def\provide@command#1{%
9533   \begingroup
9534     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9535   \endgroup
9536   \expandafter\@ifundefined\@gtempa
9537     {\def\reserved@a{\new@command#1}}%
9538     {\let\reserved@a\relax
9539     \def\reserved@a{\new@command\reserved@a}}%
9540   \reserved@a}%
9541 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9542 \def\declare@robustcommand#1{%
9543   \edef\reserved@a{\string#1}%
9544   \def\reserved@b{#1}%
9545   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9546   \edef#1{%
9547     \ifx\reserved@a\reserved@b

```

```

9548      \noexpand\x@protect
9549      \noexpand#1%
9550      \fi
9551      \noexpand\protect
9552      \expandafter\noexpand\csname
9553      \expandafter@gobble\string#1 \endcsname
9554      }%
9555      \expandafter\new@command\csname
9556      \expandafter@gobble\string#1 \endcsname
9557  }
9558  \def\x@protect#1{%
9559      \ifx\protect@typeset@protect\else
9560          \@xprotect#1%
9561      \fi
9562  }
9563  \catcode`\&=\z@ % Trick to hide conditionals
9564  \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`.

```

9565  \def\bbl@tempa{\csname newif\endcsname&ifin@}
9566  \catcode`\&=4
9567  \ifx\in@\@undefined
9568      \def\in@#1#2{%
9569          \def\in@##1#1##2##3\in@{%
9570              \ifx\in@##2\in@false\else\in@true\fi}%
9571              \in@##1\in@\in@}
9572  \else
9573      \let\bbl@tempa\@empty
9574  \fi
9575  \bbl@tempa

```

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

```

9576  \def\@ifpackagewith#1#2#3#4{#3}

```

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

```

9577  \def\@ifl@aded#1#2#3#4{}

```

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

```

9578  \ifx\@tempcnta\@undefined
9579      \csname newcount\endcsname\@tempcnta\relax
9580  \fi
9581  \ifx\@tempcntb\@undefined
9582      \csname newcount\endcsname\@tempcntb\relax
9583  \fi

```

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

```

9584  \ifx\bye\@undefined
9585      \advance\count10 by -2\relax
9586  \fi
9587  \ifx\@ifnextchar\@undefined
9588      \def\@ifnextchar#1#2#3{%
9589          \let\reserved@d=#1%
9590          \def\reserved@a{#2}\def\reserved@b{#3}%
9591          \futurelet\@let@token\@ifnch}

```

```

9592 \def\@ifnch{%
9593   \ifx\@let@token\@sptoken
9594     \let\reserved@c\@xifnch
9595   \else
9596     \ifx\@let@token\reserved@d
9597       \let\reserved@c\reserved@a
9598     \else
9599       \let\reserved@c\reserved@b
9600     \fi
9601   \fi
9602   \reserved@c}
9603 \def\:\let\@sptoken= } \: % this makes \@sptoken a space token
9604 \def\:\@xifnch} \expandafter\def\:\futurelet\@let@token\@ifnch}
9605 \fi
9606 \def\@testopt#1#2{%
9607   \ifnextchar[#{1}{#1[#2]}
9608 \def\@protected@testopt#1{%
9609   \ifx\protect\@typeset@protect
9610     \expandafter\@testopt
9611   \else
9612     \@x@protect#1%
9613   \fi}
9614 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9615   #2\relax}\fi}
9616 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9617   \else\expandafter\@gobble\fi{#1}}

```

15.4. Encoding related macros

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

```

9618 \def\DeclareTextCommand{%
9619   \@dec@text@cmd\providecommand
9620 }
9621 \def\ProvideTextCommand{%
9622   \@dec@text@cmd\providecommand
9623 }
9624 \def\DeclareTextSymbol#1#2#3{%
9625   \@dec@text@cmd\chardef#1{#2}#3\relax
9626 }
9627 \def\@dec@text@cmd#1#2#3{%
9628   \expandafter\def\expandafter#2%
9629     \expandafter{%
9630       \csname#3-cmd\expandafter\endcsname
9631       \expandafter#2%
9632       \csname#3\string#2\endcsname
9633     }%
9634 %   \let\@ifdefinable\@rc@ifdefinable
9635   \expandafter#1\csname#3\string#2\endcsname
9636 }
9637 \def\@current@cmd#1{%
9638   \ifx\protect\@typeset@protect\else
9639     \noexpand#1\expandafter\@gobble
9640   \fi
9641 }
9642 \def\@changed@cmd#1#2{%
9643   \ifx\protect\@typeset@protect
9644     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9645       \expandafter\ifx\csname ?\string#1\endcsname\relax
9646         \expandafter\def\csname ?\string#1\endcsname{%
9647           \@changed@x@err{#1}%
9648         }%
9649       \fi
9650     \global\expandafter\let

```

```

9651         \csname\cf@encoding \string#1\expandafter\endcsname
9652         \csname ?\string#1\endcsname
9653     \fi
9654     \csname\cf@encoding\string#1%
9655         \expandafter\endcsname
9656 \else
9657     \noexpand#1%
9658 \fi
9659 }
9660 \def\@changed@x@err#1{%
9661     \errhelp{Your command will be ignored, type <return> to proceed}%
9662     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9663 \def\DeclareTextCommandDefault#1{%
9664     \DeclareTextCommand#1?%
9665 }
9666 \def\ProvideTextCommandDefault#1{%
9667     \ProvideTextCommand#1?%
9668 }
9669 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9670 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9671 \def\DeclareTextAccent#1#2#3{%
9672     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
9673 }
9674 \def\DeclareTextCompositeCommand#1#2#3#4{%
9675     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9676     \edef\reserved@b{\string##1}%
9677     \edef\reserved@c{%
9678         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9679     \ifx\reserved@b\reserved@c
9680         \expandafter\expandafter\expandafter\ifx
9681             \expandafter\@car\reserved@a\relax\relax\@nil
9682             \@text@composite
9683     \else
9684         \edef\reserved@b##1{%
9685             \def\expandafter\noexpand
9686                 \csname#2\string#1\endcsname###1{%
9687                 \noexpand\@text@composite
9688                     \expandafter\noexpand\csname#2\string#1\endcsname
9689                     ###1\noexpand\@empty\noexpand\@text@composite
9690                     {##1}%
9691             }%
9692         }%
9693         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9694     \fi
9695     \expandafter\def\csname\expandafter\string\csname
9696         #2\endcsname\string#1-\string#3\endcsname{#4}
9697 \else
9698     \errhelp{Your command will be ignored, type <return> to proceed}%
9699     \errmessage{\string\DeclareTextCompositeCommand\space used on
9700         inappropriate command \protect#1}
9701 \fi
9702 }
9703 \def\@text@composite#1#2#3\@text@composite{%
9704     \expandafter\@text@composite@x
9705         \csname\string#1-\string#2\endcsname
9706 }
9707 \def\@text@composite@x#1#2{%
9708     \ifx#1\relax
9709         #2%
9710     \else
9711         #1%
9712     \fi
9713 }

```

```

9714 %
9715 \def\@strip@args#1:#2-#3\@strip@args{#2}
9716 \def\DeclareTextComposite#1#2#3#4{%
9717   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9718   \bgroup
9719     \lccode`\@=#4%
9720     \lowercase{%
9721   \egroup
9722   \reserved@a @%
9723 }%
9724 }
9725 %
9726 \def\UseTextSymbol#1#2{#2}
9727 \def\UseTextAccent#1#2#3{}
9728 \def\@use@text@encoding#1{}
9729 \def\DeclareTextSymbolDefault#1#2{%
9730   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9731 }
9732 \def\DeclareTextAccentDefault#1#2{%
9733   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9734 }
9735 \def\cf@encoding{OT1}

```

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

```

9736 \DeclareTextAccent{"}{OT1}{127}
9737 \DeclareTextAccent{'}{OT1}{19}
9738 \DeclareTextAccent{^}{OT1}{94}
9739 \DeclareTextAccent{\`}{OT1}{18}
9740 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9741 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9742 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
9743 \DeclareTextSymbol{\textquoteleft}{OT1}{``}`
9744 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
9745 \DeclareTextSymbol{\i}{OT1}{16}
9746 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9747 \ifx\scriptsize\undefined
9748   \let\scriptsize\sevenrm
9749 \fi

```

And a few more “dummy” definitions.

```

9750 \def\language{english}%
9751 \let\bbl@opt@shorthands\@nnil
9752 \def\bbl@ifshorthand#1#2#3#2{%
9753   \let\bbl@language@opts\@empty
9754   \let\bbl@provide@locale\relax
9755   \ifx\babeloptionstrings\undefined
9756     \let\bbl@opt@strings\@nnil
9757   \else
9758     \let\bbl@opt@strings\babeloptionstrings
9759   \fi
9760 \def\BabelStringsDefault{generic}
9761 \def\bbl@tempa{normal}
9762 \ifx\babeloptionmath\bbl@tempa
9763   \def\bbl@mathnormal{\noexpand\textormath}
9764 \fi
9765 \def\AfterBabelLanguage#1#2{}
9766 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9767 \let\bbl@afterlang\relax

```

```

9768 \def\bbl@opt@safe{BR}
9769 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
9770 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
9771 \expandafter\newif\csname ifbbl@single\endcsname
9772 \chardef\bbl@bidimode\z@
9773 \let\bbl@ifmetadata\gobble
9774 {(/Emulate LaTeX)}

```

A proxy file:

```

9775 {*plain}
9776 \input babel.def
9777 (/plain)

```

16. Acknowledgements

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

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

Barbara Beeton has helped in improving the manual.

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

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national $\TeX$ styles*, *TUGboat* 10 (1989) #3, pp. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *$\TeX$, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German $\TeX$* , *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International $\TeX$ is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using $\TeX$* , Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).