

Babel

Code

Version 26.11
2026/08/21

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)	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	More on selection	24
4.4	Short tags	26
4.5	Compatibility with language.def	26
4.6	Hooks	27
4.7	Setting up language files	27
4.8	Shorthands	30
4.9	Language attributes	39
4.10	Support for saving and redefining macros	40
4.11	French spacing	41
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	51
4.15.3	Shorthands for quotation marks	52
4.15.4	Umlauts and tremas	53
4.16	Layout	54
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	75
4.24	Casing	77
4.25	Getting info	78
4.26	BCP 47 related commands	79
5	Adjusting the Babel behavior	80
5.1	Cross referencing macros	82
5.2	Layout	85
5.3	Marks	86
5.4	Other packages	87
5.4.1	ifthen	87
5.4.2	varioref	88
5.4.3	hhline	88
5.5	Encoding and fonts	89
5.6	Basic bidi support	91
5.7	Local Language Configuration	94
5.8	Language options	94

6	The kernel of Babel	98
7	Error messages	99
8	Loading hyphenation patterns	103
9	luatex + xetex: common stuff	107
10	Hooks for XeTeX and LuaTeX	110
10.1	XeTeX	110
10.2	Support for interchar	112
10.3	Layout	114
10.4	8-bit TeX	115
10.5	LuaTeX	116
10.6	Southeast Asian scripts	123
10.7	CJK line breaking	124
10.8	Arabic justification	126
10.9	Common stuff	130
10.10	Automatic fonts and ids switching	131
10.11	Bidi	138
10.12	Layout	140
10.13	Lua: transforms	149
10.14	Lua: Auto bidi with basic and basic-r	159
11	Data for CJK	170
12	The ‘nil’ language	171
13	Calendars	172
13.1	Islamic	172
13.2	Hebrew	174
13.3	Persian	178
13.4	Coptic and Ethiopic	178
13.5	Julian	179
13.6	Buddhist	179
14	Support for Plain T_EX (plain.def)	181
14.1	Not renaming hyphen.tex	181
14.2	Emulating some L ^A T _E X features	182
14.3	General tools	182
14.4	Encoding related macros	186
15	Acknowledgements	188

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

1. Identification and loading of required files

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files.

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with `<@name@>` at the appropriate places in the source code and defined with either `<<name=value>>`, or with a series of lines between `<<*name>>` and `<</name>>`. The latter is cumulative (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=26.11>>
2 <<date=2026/08/21>>
```

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

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

```

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

```

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

```

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

```

\bbl@afterelse

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

3.1. A few core definitions

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

```

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

```

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

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

```

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

```

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

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

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

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

```

223 <*package>
224 \NeedsTeXFormat{LaTeX2e}
225 \ProvidesPackage{babel}%
226 [<@date@> v<@version@>
227   The multilingual framework for 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).

```

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

```

```

243 \fi}
244 % Temporary:
245 \newif\ifBabelDebugGerman
246 \@ifpackagewith{babel}{debug-german}
247 {\BabelDebugGermantrue}
248 {\BabelDebugGermanfalse}

```

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

```

249 \def\bbl@error#1{% Implicit #2#3#4
250 \begingroup
251 \catcode\`=\0 \catcode\`==12 \catcode\`'=12
252 \input errbabel.def
253 \endgroup
254 \bbl@error{#1}}
255 \def\bbl@warning#1{%
256 \begingroup
257 \def\{\{MessageBreak}%
258 \PackageWarning{babel}{#1}%
259 \endgroup}
260 \def\bbl@infowarn#1{%
261 \begingroup
262 \def\{\{MessageBreak}%
263 \PackageNote{babel}{#1}%
264 \endgroup}
265 \def\bbl@info#1{%
266 \begingroup
267 \def\{\{MessageBreak}%
268 \PackageInfo{babel}{#1}%
269 \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.

```

270 <@Basic macros>
271 \@ifpackagewith{babel}{silent}
272 {\let\bbl@info\@gobble
273 \let\bbl@infowarn\@gobble
274 \let\bbl@warning\@gobble}
275 {}
276 %
277 \def\AfterBabelLanguage#1{%
278 \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.

```

279 \ifx\bbl@languages\undefined\else
280 \begingroup
281 \catcode\`^^I=12
282 \@ifpackagewith{babel}{showlanguages}{%
283 \begingroup
284 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
285 \wlog{<*languages>}%
286 \bbl@languages
287 \wlog{</languages>}%
288 \endgroup}{%
289 \endgroup
290 \def\bbl@elt#1#2#3#4{%
291 \ifnum#2=z@
292 \gdef\bbl@nulllanguage{#1}%
293 \def\bbl@elt##1##2##3##4{%
294 \fi}%
295 \bbl@languages
296 \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`.

```
297 \bbl@trace{Defining option 'base'}
298 \@ifpackagewith{babel}{base}{%
299   \let\bbl@onlyswitch\@empty
300   \let\bbl@provide@locale\relax
301   \input babel.def
302   \let\bbl@onlyswitch\@undefined
303   \ifx\directlua\@undefined
304     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
305   \else
306     \input luababel.def
307     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
308   \fi
309   \DeclareOption{base}{}%
310   \DeclareOption{showlanguages}{}%
311   \ProcessOptions
312   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
313   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
314   \global\let\@ifl@ter@@\@ifl@ter
315   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
316   \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`.

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

```

347 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
348 \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.

```

349 \DeclareOption{KeepShorthandsActive}{}
350 \DeclareOption{activeacute}{}
351 \DeclareOption{activegrave}{}
352 \DeclareOption{debug}{}
353 \DeclareOption{debug-german}{} % Temporary
354 \DeclareOption{noconfigs}{}
355 \DeclareOption{showlanguages}{}
356 \DeclareOption{silent}{}
357 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
358 \chardef\bbl@iniflag\z@
359 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
360 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
361 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
362 \chardef\bbl@ldfflag\z@
363 \DeclareOption{provide=!}{\chardef\bbl@ldfflag\@ne} % main = 1
364 \DeclareOption{provide+=!}{\chardef\bbl@ldfflag\tw@} % second = 2
365 \DeclareOption{provide*=!}{\chardef\bbl@ldfflag\thr@@} % second + main
366 \DeclareOption{licr=extended}{}
367 \DeclareOption{licr=unextended}{}
368 % Don't use. Experimental.
369 \newif\ifbbl@single
370 \DeclareOption{selectors=off}{\bbl@singletrue}
371 <@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.

```

372 \let\bbl@opt@shorthands\@nnil
373 \let\bbl@opt@config\@nnil
374 \let\bbl@opt@main\@nnil
375 \let\bbl@opt@headfoot\@nnil
376 \let\bbl@opt@layout\@nnil
377 \let\bbl@opt@provide\@nnil

```

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

```

378 \def\bbl@tempa#1=#2\bbl@tempa{%
379   \bbl@csarg@ifx{opt#1}\@nnil
380   \bbl@csarg\edef{opt#1}{#2}%
381   \else
382   \bbl@error{bad-package-option}{#1}{#2}{}%
383   \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.

```

384 \let\bbl@language@opts\@empty
385 \DeclareOption*{%
386   \bbl@xin@{\string=}{\CurrentOption}%
387   \ifin@
388   \expandafter\bbl@tempa\CurrentOption\bbl@tempa
389   \else
390   \bbl@add@list\bbl@language@opts{\CurrentOption}%
391   \fi}

```

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

```

392 \ProcessOptions*

```

3.5. Post-process some options

```
393 \ifx\bbl@opt@provide\@nnil
394 \let\bbl@opt@provide\@empty %%% MOVE above
395 \else
396 \chardef\bbl@iniflag\@ne
397 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
398 \in@{,provide,}{, #1,}%
399 \ifin@
400 \def\bbl@opt@provide{#2}%
401 \fi}
402 \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=...`

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

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

```
417 \def\bbl@ifshorthand#1{%
418 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
419 \ifin@
420 \expandafter\@firstoftwo
421 \else
422 \expandafter\@secondoftwo
423 \fi}
```

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

```
424 \edef\bbl@opt@shorthands{%
425 \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.

```
426 \bbl@ifshorthand{'}%
427 {\PassOptionsToPackage{activeacute}{babel}}{}
428 \bbl@ifshorthand{`}%
429 {\PassOptionsToPackage{activegrave}{babel}}{}
430 \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.

```
431 \ifx\bbl@opt@headfoot\@nnil\else
432 \g@addto@macro\@resetactivechars{%
433 \set@typeset@protect
434 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
435 \let\protect\noexpand}
436 \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`.

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

```

438 \def\bbl@opt@safe{BR}
439 % \let\bbl@opt@safe\empty % Pending of \cite
440 \fi

For layout an auxiliary macro is provided, available for packages and language styles.
Optimization: if there is no layout, just do nothing.
441 \bbl@trace{Defining IfBabelLayout}
442 \ifx\bbl@opt@layout\@nnil
443 \newcommand\IfBabelLayout[3]{#3}%
444 \else
445 \bbl@exp{\bbl@forkv{\@nameuse{raw@opt@babel.sty}}}{%
446 \in{,layout,}{, #1,}%
447 \ifin@
448 \def\bbl@opt@layout{#2}%
449 \bbl@replace\bbl@opt@layout{ }{.}%
450 \fi}
451 \newcommand\IfBabelLayout[1]{%
452 \@expandtwoargs\in{.#1.}{.\bbl@opt@layout.}%
453 \ifin@
454 \expandafter\@firstoftwo
455 \else
456 \expandafter\@secondoftwo
457 \fi}
458 \fi
459 </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.

```

460 < *core >
461 \ifx\ldf@quit\undefined\else
462 \endinput\fi % Same line!
463 <@Make sure ProvidesFile is defined@>
464 \ProvidesFile{babel.def}[<@date@> v<@version@> Babel common definitions]
465 \ifx\AtBeginDocument\undefined
466 <@Emulate LaTeX@>
467 \fi
468 <@Basic macros@>
469 </core>

```

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

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

```

470 < *package | core >
471 \def\bbl@version{<@version@>}
472 \def\bbl@date{<@date@>}
473 <@Define core switching macros@>

```

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

```

474 \def\adddialect#1#2{%
475 \global\chardef#1#2\relax
476 \bbl@usehooks{adddialect}{#1}{#2}%
477 \begingroup
478 \count@#1\relax
479 \def\bbl@elt##1##2##3##4{%
480 \ifnum\count@=##2\relax
481 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
482 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'

```

```

483             set to \expandafter\string\csname l@##1\endcsname\\%
484             (\string\language\the\count@). Reported}%
485     \def\bbl@elt###1###2###3###4{}%
486     \fi}%
487     \bbl@cs{languages}%
488 \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.

```

489 \def\bbl@fixname#1{%
490   \begingroup
491   \def\bbl@tempe{l}%
492   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
493   \bbl@tempd
494     {\lowercase\expandafter{\bbl@tempd}%
495     {\uppercase\expandafter{\bbl@tempd}%
496     \@empty
497     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
498     {\uppercase\expandafter{\bbl@tempd}}}%
499     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
500     {\lowercase\expandafter{\bbl@tempd}}}%
501     \@empty
502   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
503   \bbl@tempd
504   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
505 \def\bbl@iflanguage#1{%
506   \@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).

```

507 \def\bbl@cntchars#1{\bbl@cntchars@i#1876543210\@{
508 \def\bbl@cntchars@i#1#2#3#4#5#6#7#8#9\@{\@car#9\@nil}%
509 \def\bbl@bcplookup#1{%
510   \let\bbl@templ\@empty % language
511   \let\bbl@temps\@empty % script
512   \let\bbl@tempr\@empty % regions
513   \let\bbl@tempv\@empty % variant
514   \edef\bbl@tempa{#1}%
515   \bbl@replace\bbl@tempa{-}{,}%
516   \bbl@replace\bbl@tempa{_}{,}%
517   \bbl@foreach\bbl@tempa{%
518     \ifx\bbl@templ\@empty
519       \lowercase{\def\bbl@templ{##1}}%
520     \else\ifnum\bbl@cntchars{##1}=4
521       \bbl@xin{\@car##1\@nil}{0123456789}%
522       \ifin@
523         \def\bbl@tempv{##1}% eg, 1901
524       \else
525         \uppercase{\edef\bbl@tempb{\@car##1\@nil}}%
526         \lowercase{\edef\bbl@tempb{\bbl@tempb\@gobble#1}}%
527       \fi
528     \else\ifnum\bbl@cntchars{##1}<4
529       \uppercase{\def\bbl@tempr{##1}}%

```

```

530 \else\ifnum\bb@cntchars{##1}>4
531 \lowercase{\def\bb@tempv{##1}}%
532 \fi\fi\fi\fi}%
533 \bb@exp{\bb@bcpllookup{i\bb@templ}{\bb@temps}{\bb@tempr}{\bb@tempv}}%
534 \def\bb@bcpllookup@try#1{%
535 \ifx\bb@bcpl\relax
536 \IfFileExists{babel-#1\bb@tempe.ini}{\edef\bb@bcpl{#1\bb@tempe}}{}%
537 \fi}
538 \def\bb@bcpllookup@i#1#2#3#4{%
539 \let\bb@bcpl\relax
540 \def\bb@tempe{#4}%
541 \ifx\bb@tempe\@empty\else
542 \edef\bb@tempe{-#4}%
543 \fi
544 \edef\bb@tempa{#2#3\bb@tempe}% en
545 \ifx\bb@tempa\@empty
546 \bb@bcpllookup@try{#1}%
547 \else
548 \edef\bb@tempa{#3\bb@tempe}% en-Latn
549 \ifx\bb@tempa\@empty
550 \bb@bcpllookup@try{#1-#2}%
551 \bb@bcpllookup@try{#1}%
552 \else
553 \edef\bb@tempa{#2\bb@tempe}% en-US, en-001
554 \ifx\bb@tempa\@empty
555 \bb@bcpllookup@try{#1-#3}%
556 \bb@bcpllookup@try{#1}%
557 \else % en-Latn-US
558 \bb@bcpllookup@try{#1-#2-#3}%
559 \bb@bcpllookup@try{#1-#2}%
560 \bb@bcpllookup@try{#1-#3}%
561 \bb@bcpllookup@try{#1}%
562 \fi
563 \fi
564 \fi
565 \ifx\bb@bcpl\relax
566 \ifx\bb@tempe\@empty\else\bb@bcpllookup@i{#1}{#2}{#3}{\fi}%
567 \fi}
568 %
569 \let\bb@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.

```

570 \def\iflanguage#1{%
571 \bb@iflanguage{#1}{%
572 \ifnum\csname l@#1\endcsname=\language
573 \expandafter\@firstoftwo
574 \else
575 \expandafter\@secondoftwo
576 \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.

```

577 \let\bb@select@type\z@
578 \edef\selectlanguage{%
579 \noexpand\protect
580 \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`.

```
581 \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.

```
582 \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.

```
583 \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:

```
584 \def\bbl@push@language{%
585   \ifx\language\@undefined\else
586     \ifx\currentgrouplevel\@undefined
587       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
588     \else
589       \ifnum\currentgrouplevel=\z@
590         \xdef\bbl@language@stack{\language+}%
591       \else
592         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
593       \fi
594     \fi
595   \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`.

```
596 \def\bbl@pop@lang#1+#2\@@{%
597   \edef\language{#1}%
598   \xdef\bbl@language@stack{#2}}
```

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

```
599 \let\bbl@ifrestoring\@secondoftwo
600 \def\bbl@pop@language{%
601   \expandafter\bbl@pop@lang\bbl@language@stack\@@
602   \let\bbl@ifrestoring\@firstoftwo
603   \expandafter\bbl@set@language\expandafter{\language}%
604   \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).

```

605 \chardef\localeid\z@
606 \gdef\bbl@id@last{0} % No real need for a new counter
607 \def\bbl@id@assign{%
608   \bbl@ifunset\bbl@id@\language\language}%
609   {\count@\bbl@id@last\relax
610    \advance\count@\@ne
611    \global\bbl@csarg\chardef{id@\language\language}\count@
612    \xdef\bbl@id@last{\the\count@}%
613    \ifcase\bbl@engine\or
614      \directlua{
615        Babel.locale_props[\bbl@id@last] = {}
616        Babel.locale_props[\bbl@id@last].name = '\language'
617        Babel.locale_props[\bbl@id@last].vars = {}
618      }%
619    \fi}%
620  }%
621  \chardef\localeid\bbl@cl{id@}

```

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

```

622 \let\bbl@select@opts@empty
623 \expandafter\def\csname selectlanguage \endcsname{%
624   \ifnextchar[\bbl@select@s{\bbl@select@s[]}%
625   \def\bbl@select@s[#1]#2{%
626     \def\bbl@select@opts{#1}%
627     \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw\fi
628     \bbl@push@language
629     \aftergroup\bbl@pop@language
630     \bbl@set@language{#2}%
631   \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).

```

632 \def\BabelContentsFiles{toc,lof,lot}
633 \def\bbl@set@language#1{% from selectlanguage, pop@
634   % The old buggy way. Preserved for compatibility, but simplified
635   \edef\language{\expandafter\string#1\empty}%
636   \select@language{\language}%
637   \bbl@xin@{,main,},{,\bbl@select@opts,}%
638   \ifin@
639     \let\bbl@main@language\localename
640     \let\mainlocalename\localename
641   \fi
642   \let\bbl@select@opts@empty
643   % write to aux files
644   \expandafter\ifx\csname date\language\endcsname\relax\else

```

```

645 \if@filesw
646 \bbl@xin@{,nofiles,}{,\bbl@select@opts,}%
647 \ifin@else
648 \ifx\babel@aux@gobbletwo\else % Set if single in the first, redundant
649 \bbl@savelastskip
650 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
651 \bbl@restorelastskip
652 \fi
653 \bbl@usehooks{write}{}%
654 \fi
655 \fi
656 \fi}
657 %
658 \let\bbl@restorelastskip\relax
659 \let\bbl@savelastskip\relax
660 %
661 \def\select@language#1{% from set@, babel@aux, babel@toc
662 \ifx\bbl@select@name\empty
663 \def\bbl@select@name{select}%
664 \fi
665 % set hmap
666 \ifnum\bbl@hmapsel=\@cclv\chardef\bbl@hmapsel4\relax\fi
667 % set name (when coming from babel@aux)
668 \edef\language{#1}%
669 \bbl@fixname\language
670 % define \localename when coming from set@, with a trick
671 \ifx\scantokens\undefined
672 \def\localename{??}%
673 \else
674 \bbl@exp{\scantokens{\def\localename{\language}\noexpand}\relax}%
675 \fi
676 \bbl@provide@locale
677 \bbl@iflanguage\language{%
678 \let\bbl@select@type\z@
679 \expandafter\bbl@switch\expandafter{\language}}
680 \def\babel@aux#1#2{%
681 \select@language{#1}%
682 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
683 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}%
684 \def\babel@toc#1#2{%
685 \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`.

```

686 \newif\ifbbl@usedategroup
687 \let\bbl@savextras\empty
688 \def\bbl@switch#1{% from select@, foreign@
689 % restore
690 \originalTeX
691 \expandafter\def\expandafter\originalTeX\expandafter{%
692 \csname noextras#1\endcsname

```

```

693 \let\originalTeX\@empty
694 \babel@beginsave}%
695 \bbl@usehooks{afterreset}{}%
696 \languageshorthands{none}%
697 % set the locale id
698 \bbl@id@assign
699 % switch captions, date
700 \bbl@bsphack
701 \ifcase\bbl@select@type
702 \csname captions#1\endcsname\relax
703 \csname date#1\endcsname\relax
704 \else
705 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
706 \ifin@
707 \csname captions#1\endcsname\relax
708 \fi
709 \bbl@xin@{,date,}{,\bbl@select@opts,}%
710 \ifin@ % if \foreign... within \<language>date
711 \csname date#1\endcsname\relax
712 \fi
713 \fi
714 \bbl@esphack
715 % switch extras
716 \csname bbl@preextras@#1\endcsname
717 \bbl@usehooks{beforeextras}{}%
718 \csname extras#1\endcsname\relax
719 \bbl@usehooks{afterextras}{}%
720 % > babel-ensure
721 % > babel-sh-<short>
722 % > babel-bidi
723 % > babel-fontspec
724 \let\bbl@savextras\@empty
725 % hyphenation - case mapping
726 \ifcase\bbl@opt@hyphenmap\or
727 \def\BabelLower##1##2{\lccode##1=##2\relax}%
728 \ifnum\bbl@hymap>4\else
729 \csname\language @bbl@hyphenmap\endcsname
730 \fi
731 \chardef\bbl@opt@hyphenmap\z@
732 \else
733 \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
734 \csname\language @bbl@hyphenmap\endcsname
735 \fi
736 \fi
737 \let\bbl@hymap\@cclv
738 % hyphenation - select rules
739 \ifnum\csname l@\language\endcsname=\l@unhyphenated
740 \edef\bbl@tempa{u}%
741 \else
742 \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
743 \fi
744 % linebreaking - handle u, e, k (v in the future)
745 \bbl@xin@{/u}{/\bbl@tempa}%
746 \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
747 \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
748 \ifin@ \else \bbl@xin@{/p}{/\bbl@tempa} \fi % padding (e.g., Tibetan)
749 \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
750 % hyphenation - save mins
751 \babel@savevariable\lefthyphenmin
752 \babel@savevariable\righthyphenmin
753 \ifnum\bbl@engine=@ne
754 \babel@savevariable\hyphenationmin
755 \fi

```

```

756 \ifin@
757 % unhyphenated/kashida/elongated/padding = allow stretching
758 \language\l@unhyphenated
759 \babel@savevariable\emergencystretch
760 \emergencystretch\maxdimen
761 \babel@savevariable\hbadness
762 \hbadness\@M
763 \else
764 % other = select patterns
765 \bbl@patterns{#1}%
766 \fi
767 % hyphenation - set mins
768 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
769 \set@hyphenmins\tw@\thr@@\relax
770 \@nameuse{bbl@hyphenmins@}%
771 \else
772 \expandafter\expandafter\expandafter\set@hyphenmins
773 \csname #1hyphenmins\endcsname\relax
774 \fi
775 \@nameuse{bbl@hyphenmins@}%
776 \@nameuse{bbl@hyphenmins@\language\language}%
777 \@nameuse{bbl@hyphenatmin@}%
778 \@nameuse{bbl@hyphenatmin@\language\language}%
779 \let\bbl@selectortname\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.

```

780 \edef\otherlanguage{%
781 \noexpand\protect
782 \expandafter\noexpand\csname otherlanguage \endcsname}
783 \expandafter\def\csname otherlanguage \endcsname{%
784 \@ifstar{\@nameuse{otherlanguage*}}\bbl@otherlanguage}
785 \def\bbl@otherlanguage#1{%
786 \def\bbl@selectortname{other}%
787 \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\thr@@\fi
788 \csname selectlanguage \endcsname{#1}%
789 \ignorespaces}

```

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

```

790 \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`.

```

791 \expandafter\def\csname otherlanguage*\endcsname{%
792 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
793 \def\bbl@otherlanguage@s[#1]#2{%
794 \def\bbl@selectortname{other*}%
795 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
796 \def\bbl@select@opts{#1}%
797 \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”.

```

798 \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.

```

799 \providecommand\bbl@beforeforeign{}
800 \edef\foreignlanguage{%
801   \noexpand\protect
802   \expandafter\noexpand\csname foreignlanguage \endcsname}
803 \expandafter\def\csname foreignlanguage \endcsname{%
804   \@ifstar\bbl@foreign@s\bbl@foreign@x}
805 \providecommand\bbl@foreign@x[3][]{%
806   \begingroup
807     \def\bbl@selectorname{foreign}%
808     \def\bbl@select@opts{#1}%
809     \let\BabelText\@firstofone
810     \bbl@beforeforeign
811     \foreign@language{#2}%
812     \bbl@usehooks{foreign}{}%
813     \BabelText{#3}% Now in horizontal mode!
814   \endgroup}
815 \def\bbl@foreign@s#1#2{%
816   \begingroup
817     {\par}%
818     \def\bbl@selectorname{foreign*}%
819     \let\bbl@select@opts\@empty
820     \let\BabelText\@firstofone
821     \foreign@language{#1}%
822     \bbl@usehooks{foreign*}{}%
823     \bbl@dirparastext
824     \BabelText{#2}% Still in vertical mode!
825   {\par}%
826   \endgroup}
827 \providecommand\BabelWrapText[1]{%
828   \def\bbl@tempa{\def\BabelText###1}%
829   \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`.

```

830 \def\foreign@language#1{%
831   % set name
832   \edef\language#1}%
833 \ifbbl@usedategroup
834   \bbl@add\bbl@select@opts{,date,}%
835   \bbl@usedategroupfalse
836 \fi
837 \bbl@fixname\language
838 \let\localname\language
839 \bbl@provide@locale
840 \bbl@iflanguage\language%
```

```

841 \let\bbl@select@type\@ne
842 \expandafter\bbl@switch\expandafter{\language\name}}

```

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

```

843 \def\IfBabelSelectorTF#1{%
844 \bbl@xin@{\bbl@select@name,}{,\zap@space#1 \@empty,}%
845 \ifin@
846 \expandafter\@firstoftwo
847 \else
848 \expandafter\@secondoftwo
849 \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.

```

850 \let\bbl@hyphlist\@empty
851 \let\bbl@hyphenation@\relax
852 \let\bbl@pttnlist\@empty
853 \let\bbl@patterns@\relax
854 \let\bbl@hymapsel=\@cclv
855 \def\bbl@patterns#1{%
856 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
857 \csname l@#1\endcsname
858 \edef\bbl@tempa{#1}%
859 \else
860 \csname l@#1:\f@encoding\endcsname
861 \edef\bbl@tempa{#1:\f@encoding}%
862 \fi
863 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
864 % > luatex
865 \ifundefined{bbl@hyphenation@}{% Can be \relax!
866 \begingroup
867 \bbl@xin@{\number\language,}{,\bbl@hyphlist}%
868 \ifin@\else
869 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
870 \hyphenation{%
871 \bbl@hyphenation@
872 \ifundefined{bbl@hyphenation@#1}%
873 \@empty
874 {\space\csname bbl@hyphenation@#1\endcsname}}%
875 \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
876 \fi
877 \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*.

```

878 \def\hyphenrules#1{%
879 \edef\bbl@tempf{#1}%
880 \bbl@fixname\bbl@tempf
881 \bbl@iflanguage\bbl@tempf{%
882 \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
883 \ifx\languageshorthands\undefined\else
884 \languageshorthands{none}%
885 \fi
886 \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
887 \set@hyphenmins\tw@\thr@@\relax

```

```

888 \else
889 \expandafter\expandafter\expandafter\set@hyphenmins
890 \csname\bbl@tempf hyphenmins\endcsname\relax
891 \fi}}
892 \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.

```

893 \def\providehyphenmins#1#2{%
894 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
895 \@namedef{#1hyphenmins}{#2}%
896 \fi}

```

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

```

897 \def\set@hyphenmins#1#2{%
898 \lefthyphenmin#1\relax
899 \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.

```

900 \ifx\ProvidesFile\@undefined
901 \def\ProvidesLanguage#1[#2 #3 #4]{%
902 \wlog{Language: #1 #4 #3 <#2>}%
903 }
904 \else
905 \def\ProvidesLanguage#1{%
906 \begingroup
907 \catcode`\ 10 %
908 \@makeother\/%
909 \@ifnextchar[%]
910 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
911 \def\@provideslanguage#1[#2]{%
912 \wlog{Language: #1 #2}%
913 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
914 \endgroup}
915 \fi

```

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

```

916 \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.

```

917 \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’:

```

918 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
919 \let\uselocale\setlocale
920 \let\locale\setlocale
921 \let\selectlocale\setlocale
922 \let\textlocale\setlocale
923 \let\textlanguage\setlocale
924 \let\languagetext\setlocale

```

4.2. Errors

\@nolanerr

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

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

When the format knows about `\PackageError` it must be $\text{\TeX 2}_{\epsilon}$, 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.

```

925 \edef\bbl@nulllanguage{\string\language=0}
926 \def\bbl@nocaption{\protect\bbl@nocaption@i}
927 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
928   \global\@namedef{#2}{\textbf{?#1?}}%
929   \@nameuse{#2}%
930   \edef\bbl@tempa{#1}%
931   \bbl@sreplace\bbl@tempa{name}{}%
932   \bbl@sreplace\bbl@tempa{NAME}{}%
933   \bbl@warning{%
934     \@backslashchar#1 not set for '\language'. Please,\\%
935     define it after the language has been loaded\\%
936     (typically in the preamble) with:\\%
937     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
938     Feel free to contribute on github.com/latex3/babel.\\%
939     Reported}}
940 \def\bbl@tentative{\protect\bbl@tentative@i}
941 \def\bbl@tentative@i#1{%
942   \bbl@warning{%
943     Some functions for '#1' are tentative.\\%
944     They might not work as expected and their behavior\\%
945     could change in the future.\\%
946     Reported}}
947 \def\@nolanerr#1{\bbl@error{undefined-language}{#1}{}}
948 \def\@nopatterns#1{%
949   \bbl@warning
950   {No hyphenation patterns were preloaded for\\%
951    the language '#1' into the format.\\%
952    Please, configure your TeX system to add them and\\%
953    rebuild the format. Now I will use the patterns\\%
954    preloaded for \bbl@nulllanguage\space instead}}
955 \let\bbl@usehooks\@gobbletwo

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

956 \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.

```

957 \bbl@trace{Defining babelensure}
958 \newcommand\babelensure[2][]{%
959   \AddBabelHook{babel-ensure}{afterextras}{%
960     \ifcase\bbl@select@type

```

```

961     \bbl@cl{e}%
962 \fi}%
963 \begingroup
964 \let\bbl@ens@include\@empty
965 \let\bbl@ens@exclude\@empty
966 \def\bbl@ens@fontenc{\relax}%
967 \def\bbl@tempb##1{%
968     \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
969 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
970 \def\bbl@tempb##1=##2\@@{\@namedef\bbl@ens@##1}{##2}}%
971 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
972 \def\bbl@tempc{\bbl@ensure}%
973 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
974     \expandafter\bbl@ens@include}%
975 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
976     \expandafter\bbl@ens@exclude}%
977 \toks@{\expandafter\bbl@tempc}%
978 \bbl@exp{%
979 \endgroup
980 \def<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
981 \def\bbl@ensure#1#2#3% 1: include 2: exclude 3: fontenc
982 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
983     \ifx##1\undefined % 3.32 - Don't assume the macro exists
984         \edef##1{\noexpand\bbl@nocaption
985             {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
986     \fi
987     \ifx##1\@empty\else
988         \in@{##1}{#2}%
989         \ifin@else
990             \let\bbl@tempc\language
991             \bbl@replace\bbl@tempc{-}{@}%
992             \bbl@ifunset\bbl@ensure@\bbl@tempc}%
993             {\bbl@exp{%
994                 \\DeclareRobustCommand\<\bbl@ensure@\bbl@tempc>[1]{%
995                     \\foreignlanguage{\language}%
996                     {\ifx\relax#3\else
997                         \\fontencoding{#3}\\selectfont
998                     \fi
999                     #####1}}}%
1000             }%
1001             \toks@{\expandafter{##1}%
1002             \edef##1{%
1003                 \bbl@csarg\noexpand{ensure@\bbl@tempc}%
1004                 {\the\toks@}}}%
1005         \fi
1006         \expandafter\bbl@tempb
1007     \fi}%
1008 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1009 \def\bbl@tempa##1{% elt for include list
1010     \ifx##1\@empty\else
1011         \let\bbl@tempc\language
1012         \bbl@replace\bbl@tempc{-}{@}%
1013         \bbl@csarg\in@{ensure@\bbl@tempc\expandafter}\expandafter{##1}%
1014         \ifin@else
1015             \bbl@tempb##1\@empty
1016         \fi
1017         \expandafter\bbl@tempa
1018     \fi}%
1019 \bbl@tempa#1\@empty}
1020 \def\bbl@captionslist{%
1021 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1022 \contentsname\listfigurename\listtablename\indexname\figurename
1023 \tablename\partname\enclname\ccname\headtoname\pagename\seename

```

```
1024 \alsiname\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.

```
1025 \bbl@trace{Short tags}
1026 \newcommand\babeltags[1]{%
1027   \edef\bbl@tempa{\zap@space#1 \@empty}%
1028   \def\bbl@tempb##1=##2\@{ %
1029     \edef\bbl@tempc{%
1030       \noexpand\newcommand
1031       \expandafter\noexpand\csname ##1\endcsname{%
1032         \noexpand\protect
1033         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1034       \noexpand\newcommand
1035       \expandafter\noexpand\csname text##1\endcsname{%
1036         \noexpand\foreignlanguage{##2}}
1037     \bbl@tempc}%
1038   \bbl@for\bbl@tempa\bbl@tempa{%
1039     \expandafter\bbl@tempb\bbl@tempa\@{}}
```

4.5. Compatibility with language.def

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

```
1040 \bbl@trace{Compatibility with language.def}
1041 \ifx\directlua\@undefined\else
1042   \ifx\bbl@luapatterns\@undefined
1043     \input luababel.def
1044   \fi
1045 \fi
1046 \ifx\bbl@languages\@undefined
1047   \ifx\directlua\@undefined
1048     \openin1 = language.def
1049     \ifeof1
1050       \closein1
1051       \message{I couldn't find the file language.def}
1052     \else
1053       \closein1
1054       \begingroup
1055         \def\addlanguage#1#2#3#4#5{%
1056           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1057             \global\expandafter\let\csname l@#1\expandafter\endcsname
1058             \csname lang@#1\endcsname
1059           \fi}%
1060         \def\uselanguage#1{%
1061           \input language.def
1062         \endgroup
1063       \fi
1064     \fi
1065   \chardef\l@english\zap
1066 \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.

```
1067 \def\addto#1#2{%
1068   \ifx#1\@undefined
```

```

1069 \def#1{#2}%
1070 \else
1071 \ifx#1\relax
1072 \def#1{#2}%
1073 \else
1074 {\toks@\expandafter{#1#2}%
1075 \xdef#1{\the\toks@}}%
1076 \fi
1077 \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.

```

1078 \bbl@trace{Hooks}
1079 \newcommand\AddBabelHook[3][ ]{%
1080 \bbl@iifunset\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1081 \def\bbl@tempa##1,##3=\##2,##3\@empty{\def\bbl@tempb{##2}}%
1082 \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1083 \bbl@iifunset\bbl@ev@#2@#3@#1{%
1084 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1085 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1086 \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1087 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1088 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1089 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1090 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1091 \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1092 \def\bbl@elth##1{%
1093 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}}%
1094 \bbl@cs{ev@#2@}%
1095 \ifx\language\@undefined\else % Test required for Plain (?)
1096 \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1097 \def\bbl@elth##1{%
1098 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}}%
1099 \bbl@cs{ev@#2@#1}%
1100 \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).

```

1101 \def\bbl@evargs{,% <- don't delete this comma
1102 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1103 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1104 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1105 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1106 beforestart=0,language=2,begindocument=1}
1107 \ifx\NewHook\@undefined\else % Test for Plain (?)
1108 \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}\NewHook{babel/*/#1}}
1109 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1110 \fi

```

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

```

1111 \providecommand\PassOptionsToLocale[2]{%
1112 \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`.

```

1113 \bbl@trace{Macros for setting language files up}
1114 \def\bbl@ldfinit{%
1115   \let\bbl@screset\@empty
1116   \let\BabelStrings\bbl@opt@string
1117   \let\BabelOptions\@empty
1118   \let\BabelLanguages\relax
1119   \ifx\originalTeX\@undefined
1120     \let\originalTeX\@empty
1121   \else
1122     \originalTeX
1123   \fi}
1124 \def\LdfInit#1#2{%
1125   \chardef\atcatcode=\catcode`\@
1126   \catcode`\@=11\relax
1127   \chardef\eqcatcode=\catcode`\=
1128   \catcode`\==12\relax
1129   \ifpackagewith{babel}{ensureinfo=off}}}%
1130   {\ifx\InputIfFileExists\@undefined\else
1131     \bbl@ifunset{bbl@lname@#1}%
1132     {{\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1133       \def\language#1}%
1134       \bbl@id@assign
1135       \bbl@load@info{#1}}}%
1136     {}%
1137   \fi}%
1138   \expandafter\if\expandafter\@backslashchar
1139     \expandafter\@car\string#2\@nil
1140   \ifx#2\@undefined\else
1141     \ldf@quit{#1}%
1142   \fi
1143 \else
1144   \expandafter\ifx\csname#2\endcsname\relax\else
1145     \ldf@quit{#1}%
1146   \fi
1147 \fi
1148 \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.

```

1149 \def\ldf@quit#1{%
1150   \expandafter\main@language\expandafter{#1}%
1151   \catcode`\@=\atcatcode \let\atcatcode\relax
1152   \catcode`\==\eqcatcode \let\eqcatcode\relax
1153   \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.

```

1154 \def\bbl@afterldf{%
1155   \bbl@afterlang
1156   \let\bbl@afterlang\relax
1157   \let\BabelModifiers\relax
1158   \let\bbl@screset\relax}%
1159 \def\ldf@finish#1{%
1160   \loadlocalcfg{#1}%
1161   \bbl@afterldf
1162   \expandafter\main@language\expandafter{#1}%
1163   \catcode`\@=\atcatcode \let\atcatcode\relax
1164   \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 *ltx*.

```

1165 \@onlypreamble\LdfInit
1166 \@onlypreamble\ldf@quit
1167 \@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.

```

1168 \def\main@language#1{%
1169   \def\bbl@main@language{#1}%
1170   \let\language\name\bbl@main@language
1171   \let\localname\bbl@main@language
1172   \let\mainlocalname\bbl@main@language
1173   \bbl@id@assign
1174   \ifcase\bbl@engine\or
1175     \ifx\setattribute\undefined\else
1176       \setattribute\bbl@attr@locale\localeid
1177     \fi
1178   \fi
1179   \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.

```

1180 \def\bbl@beforestart{%
1181   \def\@nolanerr##1{%
1182     \bbl@carg\chardef{l@##1}\z@
1183     \bbl@warning{Undefined language '##1' in aux.\Reported}}%
1184   \bbl@usehooks{beforestart}}%
1185   \global\let\bbl@beforestart\relax}
1186 \AtBeginDocument{%
1187   {\@nameuse\bbl@beforestart}}% Group!
1188   \if@files
1189     \providecommand\babel@aux[2]{}%
1190     \immediate\write\@mainaux{unexpanded{%
1191       \providecommand\babel@aux[2]{\global\let\babel@toc@gobbletwo}}}%
1192     \immediate\write\@mainaux{string\@nameuse\bbl@beforestart}}%
1193   \fi
1194   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1195   \ifbbl@single % must go after the line above.
1196     \renewcommand\selectlanguage[1]{}%
1197     \renewcommand\foreignlanguage[2]{#2}%
1198     \global\let\babel@aux@gobbletwo % Also as flag
1199   \fi}

```

```

1200 %
1201 \ifcase\bbl@engine\or
1202   \AtBeginDocument{\pagedir\bodydir}
1203 \fi

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

1204 \def\select@language@x#1{%
1205   \ifcase\bbl@select@type
1206     \bbl@ifsamestring\language#1\{\}\select@language{#1}}%
1207   \else
1208     \select@language{#1}%
1209 \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.

```

1210 \bbl@trace{Shorhands}
1211 \def\bbl@withactive#1#2{%
1212   \begingroup
1213     \lccode`~=#2\relax
1214     \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.

```

1215 \def\bbl@add@special#1{% 1:a macro like "\", \?, etc.
1216   \bbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1217   \bbl@ifunset{\@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1218   \ifx\nfss@catcodes\undefined\else
1219     \begingroup
1220       \catcode`#1\active
1221       \nfss@catcodes
1222       \ifnum\catcode`#1=\active
1223         \endgroup
1224         \bbl@add\nfss@catcodes{\@makeother#1}%
1225       \else
1226         \endgroup
1227     \fi
1228 \fi}

```

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

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

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

```

1229 \def\bbl@active@def#1#2#3#4{%
1230   \@namedef{#3#1}{%
1231     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1232       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1233     \else
1234       \bbl@afterfi\csname#2@sh@#1@\endcsname
1235     \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.

```

1236   \long\@namedef{#3@arg#1}##1{%
1237     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1238       \bbl@afterelse\csname#4#1\endcsname##1%
1239     \else
1240       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1241     \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.

```

1242 \def\initiate@active@char#1{%
1243   \bbl@ifunset{active@char\string#1}%
1244   {\bbl@withactive
1245     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1246   {}}

```

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).

```

1247 \def\@initiate@active@char#1#2#3{%
1248   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1249   \ifx#1\@undefined
1250     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}}%
1251   \else
1252     \bbl@csarg\let{oridef@#2}#1%
1253     \bbl@csarg\edef{oridef@#2}{%
1254       \let\noexpand#1%
1255       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1256   \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*").

```

1257   \ifx#1#3\relax
1258     \expandafter\let\csname normal@char#2\endcsname#3%
1259   \else
1260     \bbl@info{Making #2 an active character}%
1261     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1262     \@namedef{normal@char#2}{%
1263       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}}%
1264   \else
1265     \@namedef{normal@char#2}{#3}%
1266   \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).

```

1267   \bbl@restoreactive{#2}%
1268   \AtBeginDocument{%

```

```

1269 \catcode`#2\active
1270 \if@filesw
1271 \immediate\write\@mainaux{\catcode`\string#2\active}%
1272 \fi}%
1273 \expandafter\bbbl@add@special\csname#2\endcsname
1274 \catcode`#2\active
1275 \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>`).

```

1276 \let\bbbl@tempa\@firstoftwo
1277 \if\string^#2%
1278 \def\bbbl@tempa{\noexpand\textormath}%
1279 \else
1280 \ifx\bbbl@mathnormal\@undefined\else
1281 \let\bbbl@tempa\bbbl@mathnormal
1282 \fi
1283 \fi
1284 \expandafter\edef\csname active@char#2\endcsname{%
1285 \bbbl@tempa
1286 {\noexpand\if@safe@actives
1287 \noexpand\expandafter
1288 \expandafter\noexpand\csname normal@char#2\endcsname
1289 \noexpand\else
1290 \noexpand\expandafter
1291 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1292 \noexpand\fi}%
1293 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1294 \bbbl@csarg\edef{doactive#2}{%
1295 \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!).

```

1296 \bbbl@csarg\edef{active@#2}{%
1297 \noexpand\active@prefix\noexpand#1%
1298 \expandafter\noexpand\csname active@char#2\endcsname}%
1299 \bbbl@csarg\edef{normal@#2}{%
1300 \noexpand\active@prefix\noexpand#1%
1301 \expandafter\noexpand\csname normal@char#2\endcsname}%
1302 \bbbl@ncarg\let#1\bbbl@normal@#2}%

```

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

```

1303 \bbbl@active@def#2\user@group{user@active}{language@active}%
1304 \bbbl@active@def#2\language@group{language@active}{system@active}%
1305 \bbbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1306 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1307 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1308 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1309 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (') active we need to change `\pr@ms` as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```

1310 \if\string'#2%
1311   \let\prim@s\bbl@prim@s
1312   \let\active@math@prime#1%
1313 \fi
1314 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

1315 <<{*More package options}>> ≡
1316 \DeclareOption{math=active}{}
1317 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1318 <</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*.

```

1319 \@ifpackagewith{babel}{KeepShorthandsActive}%
1320 {\let\bbl@restoreactive\@gobble}%
1321 {\def\bbl@restoreactive#1{%
1322   \bbl@exp{%
1323     \\AfterBabelLanguage\\CurrentOption
1324     {\catcode`#1=\the\catcode`#1\relax}%
1325     \\AtEndOfPackage
1326     {\catcode`#1=\the\catcode`#1\relax}}}%
1327   \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.

```

1328 \def\bbl@sh@select#1#2{%
1329   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1330     \bbl@afterelse\bbl@scndcs
1331   \else
1332     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1333   \fi}

```

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

```

1334 \begingroup
1335 \bbl@ifunset{ifincsname}
1336 {\gdef\active@prefix#1{%
1337   \ifx\protect\@typeset@protect
1338   \else
1339     \ifx\protect\@unexpandable@protect
1340       \noexpand#1%
1341     \else
1342       \protect#1%
1343     \fi
1344     \expandafter\@gobble
1345   \fi}}
1346 {\gdef\active@prefix#1{%
1347   \ifincsname

```

```

1348     \string#1%
1349     \expandafter\@gobble
1350   \else
1351     \ifx\protect\@typeset@protect
1352     \else
1353       \ifx\protect\@unexpandable@protect
1354         \noexpand#1%
1355       \else
1356         \protect#1%
1357       \fi
1358     \expandafter\expandafter\expandafter\@gobble
1359   \fi
1360 \fi}}
1361 \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`).

```

1362 \newif\if@safe@actives
1363 \@safe@activefalse

```

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

```

1364 \def\bbl@restore@actives{\if@safe@actives\@safe@activefalse\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`.

```

1365 \chardef\bbl@activated\z@
1366 \def\bbl@activate#1{%
1367   \chardef\bbl@activated\@ne
1368   \bbl@withactive{\expandafter\let\expandafter}#1%
1369   \csname bbl@active@string#1\endcsname}
1370 \def\bbl@deactivate#1{%
1371   \chardef\bbl@activated\tw@
1372   \bbl@withactive{\expandafter\let\expandafter}#1%
1373   \csname bbl@normal@string#1\endcsname}

```

\bbl@firstcs

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

```

1374 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1375 \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.

```

1376 \def\babel@texpdf#1#2#3#4{%

```

```

1377 \ifx\texorpdfstring\undefined
1378   \textormath{#1}{#3}%
1379 \else
1380   \texorpdfstring{\textormath{#1}{#3}}{#2}%
1381   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1382 \fi}
1383 %
1384 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1385 \def\@decl@short#1#2#3\@nil#4{%
1386   \def\bbl@tempa{#3}%
1387   \ifx\bbl@tempa\@empty
1388     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1389     \bbl@ifunset{#1@sh@\string#2@}{}%
1390     {\def\bbl@tempa{#4}%
1391      \expandafter\ifx\csname#1@sh@\string#2@endcsname\bbl@tempa
1392      \else
1393        \bbl@info
1394          {Redefining #1 shorthand \string#2\\%
1395           in language \CurrentOption}%
1396      \fi}%
1397     \@namedef{#1@sh@\string#2@}{#4}%
1398   \else
1399     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1400     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1401     {\def\bbl@tempa{#4}%
1402      \expandafter\ifx\csname#1@sh@\string#2@\string#3@endcsname\bbl@tempa
1403      \else
1404        \bbl@info
1405          {Redefining #1 shorthand \string#2\string#3\\%
1406           in language \CurrentOption}%
1407      \fi}%
1408     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1409   \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.

```

1410 \def\textormath{%
1411   \ifmmode
1412     \expandafter\@secondoftwo
1413   \else
1414     \expandafter\@firstoftwo
1415   \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’.

```

1416 \def\user@group{user}
1417 \def\language@group{english}
1418 \def\system@group{system}

```

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

```

1419 \def\useshorthands{%
1420   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1421 \def\bbl@usesh@s#1{%
1422   \bbl@usesh@x
1423   {\AddBabelHook{babel-sh-\string#1}{afterextras}}{\bbl@activate{#1}}}%
1424   {#1}}

```

```

1425 \def\bbl@usesh@x#1#2{%
1426   \bbl@ifshorthand{#2}%
1427   {\def\user@group{user}%
1428     \initiate@active@char{#2}%
1429     #1%
1430     \bbl@activate{#2}}%
1431   {\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.

```

1432 \def\user@language@group{user@\language@group}
1433 \def\bbl@set@user@generic#1#2{%
1434   \bbl@ifunset{user@generic@active#1}%
1435   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1436     \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1437     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1438       \expandafter\noexpand\csname normal@char#1\endcsname}%
1439     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1440       \expandafter\noexpand\csname user@active#1\endcsname}%
1441     \@empty}
1442   \newcommand\defineshorthand[3][user]{%
1443     \edef\bbl@tempa{\zap@space#1 \@empty}%
1444     \bbl@for\bbl@tempb\bbl@tempa{%
1445       \if*\expandafter\@car\bbl@tempb\@nil
1446       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1447       \@expandtwoargs
1448       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1449     }
1450     \fi
1451     \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.

```

1451 \def\languageshorthands#1{%
1452   \bbl@ifsamestring{none}{#1}{}%
1453   \bbl@once{short-\localename-#1}{%
1454     \bbl@info{'\localename' activates '#1' shorthands.\Reported}}}%
1455   \def\language@group{#1}}

```

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

```

1456 \def\aliasshorthand#1#2{%
1457   \bbl@ifshorthand{#2}%
1458   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1459     \ifx\document@notprerr
1460       \@notshorthand{#2}%
1461     \else
1462       \initiate@active@char{#2}%
1463       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1464       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1465       \bbl@activate{#2}%
1466     \fi
1467   \fi}%
1468   {\bbl@error{shorthand-is-off}{#2}{}}}

```

\@notshorthand

```

1469 \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.

```
1470 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1471 \DeclareRobustCommand*\shorthandoff{%
1472   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1473 \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.

```
1474 \def\bbl@switch@sh#1#2{%
1475   \ifx#2\@nnil\else
1476     \bbl@ifunset{bbl@active@\string#2}%
1477     {\bbl@error{not-a-shorthand-b}{\string#2}}}%
1478     {\ifcase#1%   off, on, off*
1479       \catcode`#2\relax
1480       \or
1481       \catcode`#2\active
1482       \bbl@ifunset{bbl@shdef@\string#2}%
1483       {}%
1484       {\bbl@withactive{\expandafter\let\expandafter}#2%
1485         \csname bbl@shdef@\string#2\endcsname
1486         \bbl@csarg\let{shdef@\string#2}\relax}%
1487       \ifcase\bbl@activated\or
1488         \bbl@activate{#2}%
1489       \else
1490         \bbl@deactivate{#2}%
1491       \fi
1492       \or
1493       \bbl@ifunset{bbl@shdef@\string#2}%
1494       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1495       {}%
1496       \csname bbl@oricat@\string#2\endcsname
1497       \csname bbl@oridef@\string#2\endcsname
1498       \fi}%
1499   \bbl@afterfi\bbl@switch@sh#1%
1500 \fi}
```

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

```
1501 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1502 \def\bbl@putsh#1{%
1503   \bbl@ifunset{bbl@active@\string#1}%
1504   {\bbl@putsh@i#1\@empty\@nnil}%
1505   {\csname bbl@active@\string#1\endcsname}}
1506 \def\bbl@putsh@i#1#2\@nnil{%
1507   \csname\language@group @sh@\string#1@%
1508     \ifx\@empty#2\else\string#2@\fi\endcsname}
1509 %
1510 \ifx\bbl@opt@shorthands\@nnil\else
1511   \let\bbl@s@initiate@active@char\initiate@active@char
1512   \def\initiate@active@char#1{%
1513     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1514   \let\bbl@s@switch@sh\bbl@switch@sh
1515   \def\bbl@switch@sh#1#2{%
1516     \ifx#2\@nnil\else
```

```

1517 \bbl@afterfi
1518 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1519 \fi}
1520 \let\bbl@s@activate\bbl@activate
1521 \def\bbl@activate#1{%
1522 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1523 \let\bbl@s@deactivate\bbl@deactivate
1524 \def\bbl@deactivate#1{%
1525 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1526 \fi

```

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

```

1527 \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.

```

1528 \def\bbl@prim@s{%
1529 \prime\futurelet\@let@token\bbl@pr@m@s}
1530 \def\bbl@if@primes#1#2{%
1531 \ifx#1\@let@token
1532 \expandafter\@firstoftwo
1533 \else\ifx#2\@let@token
1534 \bbl@afterelse\expandafter\@firstoftwo
1535 \else
1536 \bbl@afterfi\expandafter\@secondoftwo
1537 \fi\fi}
1538 \begingroup
1539 \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1540 \catcode`\'=12 \catcode`\"=\active \lccode`\"='\ '
1541 \lowercase{%
1542 \gdef\bbl@pr@m@s{%
1543 \bbl@if@primes" '%
1544 \pr@@@s
1545 {\bbl@if@primes*\^pr@@@t\egroup}}}
1546 \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).

```

1547 \initiate@active@char{~}
1548 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1549 \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.

```

1550 \expandafter\def\csname OT1dqpos\endcsname{127}
1551 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

1552 \ifx\f@encoding\undefined
1553 \def\f@encoding{OT1}
1554 \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.

```
1555 \bbl@trace{Language attributes}
1556 \newcommand\languageattribute[2]{%
1557   \def\bbl@tempc{#1}%
1558   \bbl@fixname\bbl@tempc
1559   \bbl@iflanguage\bbl@tempc{%
1560     \bbl@vforeach{#2}{%
```

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

```
1561     \ifx\bbl@known@attrs\undefined
1562       \in@false
1563     \else
1564       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1565     \fi
1566     \ifin@
1567       \bbl@warning{%
1568         You have more than once selected the attribute '##1'\%
1569         for language #1. Reported}%
1570     \else
```

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

```
1571       \bbl@info{Activated '##1' attribute for\%
1572         '\bbl@tempc'. Reported}%
1573       \bbl@exp{%
1574         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1575       \edef\bbl@tempa{\bbl@tempc-##1}%
1576       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1577       {\csname\bbl@tempc @attr##1\endcsname}%
1578       {\@attrerr{\bbl@tempc}{##1}}%
1579     \fi}}
1580 \@onlypreamble\languageattribute
```

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

```
1581 \newcommand*\@attrerr[2]{%
1582   \bbl@error{unknown-attribute}{#1}{#2}{}}
```

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

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

```
1583 \def\bbl@declare@ttribute#1#2#3{%
1584   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1585   \ifin@
1586     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1587   \fi
1588   \bbl@add@list\bbl@attributes{#1-#2}%
1589   \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.

```

1590 \def\bbl@ifattributeset#1#2#3#4{%
1591   \ifx\bbl@known@attrs\@undefined
1592     \in@false
1593   \else
1594     \bbl@xin@{,#1-#2,}\bbl@known@attrs,%
1595   \fi
1596   \ifin@
1597     \bbl@afterelse#3%
1598   \else
1599     \bbl@afterfi#4%
1600   \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.

```

1601 \def\bbl@ifknown@ttrib#1#2{%
1602   \let\bbl@tempa\@secondoftwo
1603   \bbl@loopx\bbl@tempb{#2}%
1604   \expandafter\in@\expandafter{\expandafter\bbl@tempb,}\bbl@tempa,%
1605   \ifin@
1606     \let\bbl@tempa\@firstoftwo
1607   \else
1608     \fi}%
1609   \bbl@tempa}

```

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

```

1610 \def\bbl@clear@ttribs{%
1611   \ifx\bbl@attributes\@undefined\else
1612     \bbl@loopx\bbl@tempa{\bbl@attributes}%
1613     \expandafter\bbl@clear@ttrib\bbl@tempa.%
1614   \let\bbl@attributes\@undefined
1615   \fi}
1616 \def\bbl@clear@ttrib#1-#2.{%
1617   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1618 \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.

```

1619 \bbl@trace{Macros for saving definitions}
1620 \def\babel@beginsave{\babel@savecnt\z@}

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

1621 \newcount\babel@savecnt
1622 \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.

```
1623 \def\babel@save#1{%
1624   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1625   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1626     \expandafter\expandafter,\bbl@savedextras,}%
1627   \expandafter\in@\bbl@tempa
1628   \ifin@
1629     \bbl@add\bbl@savedextras{,{#1,}}%
1630     \bbl@carg\let\babel@number\babel@savecnt#1\relax
1631     \toks@{\expandafter{\originalTeX\let#1=}}%
1632     \bbl@exp{%
1633       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1634     \advance\babel@savecnt@one
1635   \fi}
1636 \def\babel@savevariable#1{%
1637   \toks@{\expandafter{\originalTeX #1=}}%
1638   \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`.

```
1639 \def\bbl@redefine#1{%
1640   \edef\bbl@tempa{\bbl@stripslash#1}%
1641   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1642   \expandafter\def\csname\bbl@tempa\endcsname}
1643 \@onlypreamble\bbl@redefine
```

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

```
1644 \def\bbl@redefine@long#1{%
1645   \edef\bbl@tempa{\bbl@stripslash#1}%
1646   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1647   \long\expandafter\def\csname\bbl@tempa\endcsname}
1648 \@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_`.

```
1649 \def\bbl@redefineroobust#1{%
1650   \edef\bbl@tempa{\bbl@stripslash#1}%
1651   \bbl@ifunset{\bbl@tempa\space}%
1652     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1653       \bbl@exp{\def\\#1{\protect<\bbl@tempa\space>}}}%
1654     {\bbl@exp{\let<org@\bbl@tempa><\bbl@tempa\space>}}}%
1655     \@namedef{\bbl@tempa\space}}
1656 \@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.

```

1657 \def\bbl@frenchspacing{%
1658   \ifnum\the\sfcode`\.\=@m
1659     \let\bbl@nonfrenchspacing\relax
1660   \else
1661     \frenchspacing
1662     \let\bbl@nonfrenchspacing\nonfrenchspacing
1663   \fi}
1664 \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.

```

1665 \let\bbl@elt\relax
1666 \edef\bbl@fs@chars{%
1667   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1668   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1669   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1670 \def\bbl@pre@fs{%
1671   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1672   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1673 \def\bbl@post@fs{%
1674   \bbl@save@sfcodes
1675   \edef\bbl@tempa{\bbl@cl{frspc}}%
1676   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1677   \if u\bbl@tempa      % do nothing
1678   \else\if n\bbl@tempa % non french
1679     \def\bbl@elt##1##2##3{%
1680       \ifnum\sfcode`##1=##2\relax
1681       \babel@savevariable{\sfcode`##1}%
1682       \sfcode`##1=##3\relax
1683     \fi}%
1684     \bbl@fs@chars
1685   \else\if y\bbl@tempa % french
1686     \def\bbl@elt##1##2##3{%
1687       \ifnum\sfcode`##1=##3\relax
1688       \babel@savevariable{\sfcode`##1}%
1689       \sfcode`##1=##2\relax
1690     \fi}%
1691     \bbl@fs@chars
1692   \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.

```

1693 \bbl@trace{Hyphens}
1694 \@onlypreamble\babelhyphenation
1695 \AtEndOfPackage{%
1696   \newcommand\babelhyphenation[2][\@empty]{%
1697     \ifx\bbl@hyphenation@\relax
1698       \let\bbl@hyphenation@\@empty
1699     \fi
1700     \ifx\bbl@hyphlist\@empty\else
1701       \bbl@warning{%
1702         You must not intermingle \string\selectlanguage\space and\\%
1703         \string\babelhyphenation\space or some exceptions will not\\%
1704         be taken into account. Reported}%
1705       \fi

```

```

1706 \ifx\@empty#1%
1707 \protected@edef\bb@hyphenation@\bb@hyphenation\space#2}%
1708 \else
1709 \bb@vforeach{#1}{%
1710 \def\bb@tempa{##1}%
1711 \bb@fixname\bb@tempa
1712 \bb@iflanguage\bb@tempa{%
1713 \bb@csarg\protected@edef\hyphenation@\bb@tempa}{%
1714 \bb@ifunset\bb@hyphenation@\bb@tempa}%
1715 }%
1716 {\csname \bb@hyphenation@\bb@tempa\endcsname\space}%
1717 #2}}}%
1718 \fi}}

```

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

```

1719 \ifx\NewDocumentCommand\@undefined\else
1720 \NewDocumentCommand\babelhyphenmins{sommo}{%
1721 \IfNoValueTF{#2}%
1722 {\protected@edef\bb@hyphenmins@\set@hyphenmins{#3}{#4}}%
1723 \IfValueT{#5}{%
1724 \protected@edef\bb@hyphenatmin@\hyphenationmin=#5\relax}}%
1725 \IfBooleanT{#1}{%
1726 \leftthyphenmin=#3\relax
1727 \rightthyphenmin=#4\relax
1728 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1729 {\edef\bb@tempb{\zap@space#2 \@empty}%
1730 \bb@for\bb@tempa\bb@tempb{%
1731 \@namedef\bb@hyphenmins@\bb@tempa}{\set@hyphenmins{#3}{#4}}%
1732 \IfValueT{#5}{%
1733 \@namedef\bb@hyphenatmin@\bb@tempa}{\hyphenationmin=#5\relax}}}%
1734 \IfBooleanT{#1}{\bb@error{hyphenmins-args}{}}}%
1735 \fi

```

\bb@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.

```

1736 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\zap@space\fi}
1737 \def\bb@t@one{T1}
1738 \def\allowhyphens{\ifx\cf@encoding\bb@t@one\else\bb@allowhyphens\fi}

```

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

```

1739 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1740 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1741 \def\bb@hyphen{%
1742 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
1743 \def\bb@hyphen@i#1#2{%
1744 \lowercase{\bb@ifunset\bb@hy@#1#2\@empty}}%
1745 {\csname \bb@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1746 {\lowercase{\csname \bb@hy@#1#2\@empty\endcsname}}}

```

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

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

```

1747 \def\bb@usehyphen#1{%
1748 \leavevmode

```

```

1749 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1750 \nobreak\hskip\z@skip}
1751 \def\bbl@usehyphen#1{%
1752 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1753 \def\bbl@hyphenchar{%
1754 \ifnum\hyphenchar\font=\m@ne
1755 \babelnullhyphen
1756 \else
1757 \char\hyphenchar\font
1758 \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.

```

1759 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1760 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1761 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1762 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1763 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1764 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1765 \def\bbl@hy@repeat{%
1766 \bbl@usehyphen{
1767 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1768 \def\bbl@hy@@repeat{%
1769 \bbl@usehyphen{
1770 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1771 \def\bbl@hy@empty{\hskip\z@skip}
1772 \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.

```

1773 \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.

```

1774 \bbl@trace{Multiencoding strings}
1775 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1776 <<*More package options>> ≡
1777 \DeclareOption{nocase}{}
1778 <</More package options>>

```

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

```

1779 <<*More package options>> ≡
1780 \let\bbl@opt@strings\@nnil % accept strings=value
1781 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1782 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1783 \def\BabelStringsDefault{generic}
1784 <</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.

```

1785 \@onlypreamble\StartBabelCommands
1786 \def\StartBabelCommands{%
1787   \begingroup
1788   \@tempcnta="7F
1789   \def\bbl@tempa{%
1790     \ifnum\@tempcnta>"FF\else
1791       \catcode\@tempcnta=11
1792       \advance\@tempcnta\@ne
1793       \expandafter\bbl@tempa
1794     \fi}%
1795   \bbl@tempa
1796   <@Macros local to BabelCommands@>
1797   \def\bbl@provstring##1##2{%
1798     \providecommand##1{##2}%
1799     \bbl@tglobal##1}%
1800   \global\let\bbl@scafter\@empty
1801   \let\StartBabelCommands\bbl@startcmds
1802   \ifx\BabelLanguages\relax
1803     \let\BabelLanguages\CurrentOption
1804   \fi
1805   \begingroup
1806   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1807   \StartBabelCommands}
1808 \def\bbl@startcmds{%
1809   \ifx\bbl@screset\@nnil\else
1810     \bbl@usehooks{stopcommands}{}%
1811   \fi
1812   \endgroup
1813   \begingroup
1814   \@ifstar
1815     {\ifx\bbl@opt@strings\@nnil
1816       \let\bbl@opt@strings\BabelStringsDefault
1817     \fi
1818     \bbl@startcmds@i}%
1819   \bbl@startcmds@i}
1820 \def\bbl@startcmds@i##1##2{%
1821   \edef\bbl@L{\zap@space#1 \@empty}%
1822   \edef\bbl@G{\zap@space#2 \@empty}%
1823   \bbl@startcmds@ii}
1824 \let\bbl@startcommands\StartBabelCommands

```

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

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

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

```

1825 \newcommand\bbl@startcmds@ii[[1][\@empty]]{%
1826   \let\SetString\@gobbletwo
1827   \let\bbl@stringdef\@gobbletwo
1828   \let\AfterBabelCommands\@gobble
1829   \ifx\@empty#1%
1830     \def\bbl@sc@label{generic}%
1831     \def\bbl@encstring##1##2{%
1832       \ProvideTextCommandDefault##1{##2}%
1833       \bbl@tglobal##1%
1834       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1835 \let\bbl@sctest\in@true
1836 \else
1837 \let\bbl@sc@charset\space % <- zapped below
1838 \let\bbl@sc@fontenc\space % <- " "
1839 \def\bbl@tempa##1=##2\@nil{%
1840 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1841 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1842 \def\bbl@tempa##1 ##2{% space -> comma
1843 ##1%
1844 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1845 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1846 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1847 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1848 \def\bbl@encstring##1##2{%
1849 \bbl@foreach\bbl@sc@fontenc{%
1850 \bbl@ifunset{T@####1}%
1851 {}%
1852 {\ProvideTextCommand##1{####1}{##2}%
1853 \bbl@tglobal##1%
1854 \expandafter
1855 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1856 \def\bbl@sctest{%
1857 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
1858 \fi
1859 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1860 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1861 \let\AfterBabelCommands\bbl@aftercmds
1862 \let\SetString\bbl@setstring
1863 \let\bbl@stringdef\bbl@encstring
1864 \else % i.e., strings=value
1865 \bbl@sctest
1866 \ifin@
1867 \let\AfterBabelCommands\bbl@aftercmds
1868 \let\SetString\bbl@setstring
1869 \let\bbl@stringdef\bbl@provstring
1870 \fi\fi\fi
1871 \bbl@scswitch
1872 \ifx\bbl@G\@empty
1873 \def\SetString##1##2{%
1874 \bbl@error{missing-group}{##1}{}}}%
1875 \fi
1876 \ifx\@empty#1%
1877 \bbl@usehooks{defaultcommands}{}%
1878 \else
1879 \@expandtwoargs
1880 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1881 \fi}

```

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

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

```

1882 \def\bbl@forlang#1#2{%
1883 \bbl@for#1\bbl@L{%
1884 \bbl@xin@{,##1,}{,\BabelLanguages,}%
1885 \ifin@#2\relax\fi}}
1886 \def\bbl@scswitch{%
1887 \bbl@forlang\bbl@tempa{%
1888 \ifx\bbl@G\@empty\else

```

```

1889 \ifx\SetString@gobbletwo\else
1890 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1891 \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1892 \ifin@else
1893 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1894 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1895 \fi
1896 \fi
1897 \fi}}
1898 \AtEndOfPackage{%
1899 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1900 \let\bbl@scswitch\relax}
1901 \@onlypreamble\EndBabelCommands
1902 \def\EndBabelCommands{%
1903 \bbl@usehooks{stopcommands}{}%
1904 \endgroup
1905 \endgroup
1906 \bbl@scafter}
1907 \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.

```

1908 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1909 \bbl@forlang\bbl@tempa{%
1910 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1911 \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1912 {\bbl@exp{%
1913 \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1914 }%
1915 \def\BabelString{#2}%
1916 \bbl@usehooks{stringprocess}{}%
1917 \expandafter\bbl@stringdef
1918 \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.

```

1919 \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.

```

1920 << *Macros local to BabelCommands >> ≡
1921 \def\SetStringLoop##1##2{%
1922 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1923 \count@\z@
1924 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1925 \advance\count@\@ne
1926 \toks@\expandafter{\bbl@tempa}%
1927 \bbl@exp{%
1928 \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1929 \count@=\the\count@\relax}}}%
1930 << /Macros local to BabelCommands >>

```

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

```

1931 \def\bbl@aftercmds#1{%
1932 \toks@\expandafter{\bbl@scafter#1}%
1933 \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.

```

1934 <<*Macros local to BabelCommands>> ≡
1935   \newcommand\SetCase[3][]{%
1936     \def\bbl@tempa####1####2{%
1937       \ifx####1\@empty\else
1938         \bbl@carg\bbl@add{extras\CurrentOption}{%
1939           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1940           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1941           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1942           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1943         \expandafter\bbl@tempa
1944       \fi}%
1945   \bbl@tempa##1\@empty\@empty
1946   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1947 <</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.

```

1948 <<*Macros local to BabelCommands>> ≡
1949   \newcommand\SetHyphenMap[1]{%
1950     \bbl@forlang\bbl@tempa{%
1951       \expandafter\bbl@stringdef
1952       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1953 <</Macros local to BabelCommands>>

```

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

```

1954 \newcommand\BabelLower[2]{% one to one.
1955   \ifnum\lccode#1=#2\else
1956     \babel@savevariable{\lccode#1}%
1957     \lccode#1=#2\relax
1958   \fi}
1959 \newcommand\BabelLowerMM[4]{% many-to-many
1960   \@tempcnta=#1\relax
1961   \@tempcntb=#4\relax
1962   \def\bbl@tempa{%
1963     \ifnum\@tempcnta>#2\else
1964       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1965       \advance\@tempcnta#3\relax
1966       \advance\@tempcntb#3\relax
1967       \expandafter\bbl@tempa
1968     \fi}%
1969   \bbl@tempa}
1970 \newcommand\BabelLowerM0[4]{% many-to-one
1971   \@tempcnta=#1\relax
1972   \def\bbl@tempa{%
1973     \ifnum\@tempcnta>#2\else
1974       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1975       \advance\@tempcnta#3
1976       \expandafter\bbl@tempa
1977     \fi}%
1978   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1979 <<*More package options>> ≡
1980 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1981 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1982 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1983 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1984 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1985 <</More package options>>

```

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

```

1986 \AtEndOfPackage{%
1987   \ifx\bbbl@opt@hyphenmap\@undefined
1988     \bbbl@xin@{,}\bbbl@language@opts}%
1989     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1990   \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.

```

1991 \newcommand\setlocalecaption{%
1992   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1993 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1994   \bbbl@trim@def\bbbl@tempa{#2}%
1995   \bbbl@xin@{.template}\bbbl@tempa}%
1996   \ifin@
1997     \bbbl@ini@captions@template{#3}{#1}%
1998   \else
1999     \edef\bbbl@tempd{%
2000       \expandafter\expandafter\expandafter
2001       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2002     \bbbl@xin@
2003       {\expandafter\string\csname #2name\endcsname}%
2004       {\bbbl@tempd}%
2005     \ifin@ % Renew caption
2006       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
2007     \ifin@
2008       \bbbl@exp{%
2009         \\bbbl@ifsamestring{\bbbl@tempa}\language}%
2010         {\\bbbl@scset\<#2name>\<#1#2name>}%
2011         {}}%
2012       \else % Old way converts to new way
2013         \bbbl@ifunset{#1#2name}%
2014         {\bbbl@exp{%
2015           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2016           \\bbbl@ifsamestring{\bbbl@tempa}\language}%
2017           {\def\<#2name>\<#1#2name>}}%
2018           {}}}%
2019       {}%
2020     \fi
2021   \else
2022     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
2023     \ifin@ % New way
2024       \bbbl@exp{%
2025         \\bbbl@add\<captions#1>{\\bbbl@scset\<#2name>\<#1#2name>}}%
2026         \\bbbl@ifsamestring{\bbbl@tempa}\language}%
2027         {\\bbbl@scset\<#2name>\<#1#2name>}}%
2028         {}}%
2029       \else % Old way, but defined in the new way
2030       \bbbl@exp{%
2031         \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2032         \\bbbl@ifsamestring{\bbbl@tempa}\language}%
2033         {\def\<#2name>\<#1#2name>}}%
2034         {}}%
2035       \fi%
2036     \fi
2037     \@namedef{#1#2name}{#3}%
2038     \toks@ \expandafter\bbbl@captionslist}%
2039     \bbbl@exp{\in@{\<#2name>}\the\toks@}%
2040     \ifin@ \else
2041       \bbbl@exp{\\bbbl@add\\bbbl@captionslist{\<#2name>}}%

```

```

2042 \bbl@tglobal\bbl@captionslist
2043 \fi
2044 \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.

```

2045 \bbl@trace{Macros related to glyphs}
2046 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2047 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2048 \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.

```

2049 \def\save@sf@q#1{\leavevmode
2050 \begingroup
2051 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2052 \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.

```

2053 \ProvideTextCommand{\quotedblbase}{OT1}{%
2054 \save@sf@q{\set@low@box{\textquotedblright/}}%
2055 \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.

```

2056 \ProvideTextCommandDefault{\quotedblbase}{%
2057 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2058 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2059 \save@sf@q{\set@low@box{\textquoteright/}}%
2060 \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.

```

2061 \ProvideTextCommandDefault{\quotesinglbase}{%
2062 \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

2063 \ProvideTextCommand{\guillemetleft}{OT1}{%
2064 \ifmmode
2065 \ll
2066 \else
2067 \save@sf@q{\nobreak
2068 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2069 \fi}
2070 \ProvideTextCommand{\guillemetright}{OT1}{%
2071 \ifmmode
2072 \gg
2073 \else
2074 \save@sf@q{\nobreak
2075 \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%

```

```

2076 \fi}
2077 \ProvideTextCommand{\guillemotleft}{OT1}{%
2078 \ifmmode
2079 \ll
2080 \else
2081 \save@sf@q{\nobreak
2082 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2083 \fi}
2084 \ProvideTextCommand{\guillemotright}{OT1}{%
2085 \ifmmode
2086 \gg
2087 \else
2088 \save@sf@q{\nobreak
2089 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2090 \fi}

```

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

```

2091 \ProvideTextCommandDefault{\guillemetleft}{%
2092 \UseTextSymbol{OT1}{\guillemetleft}}
2093 \ProvideTextCommandDefault{\guillemetright}{%
2094 \UseTextSymbol{OT1}{\guillemetright}}
2095 \ProvideTextCommandDefault{\guillemotleft}{%
2096 \UseTextSymbol{OT1}{\guillemotleft}}
2097 \ProvideTextCommandDefault{\guillemotright}{%
2098 \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2099 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2100 \ifmmode
2101 <%
2102 \else
2103 \save@sf@q{\nobreak
2104 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2105 \fi}
2106 \ProvideTextCommand{\guilsinglright}{OT1}{%
2107 \ifmmode
2108 >%
2109 \else
2110 \save@sf@q{\nobreak
2111 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2112 \fi}

```

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

```

2113 \ProvideTextCommandDefault{\guilsinglleft}{%
2114 \UseTextSymbol{OT1}{\guilsinglleft}}
2115 \ProvideTextCommandDefault{\guilsinglright}{%
2116 \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.

```

2117 \DeclareTextCommand{\ij}{OT1}{%
2118 i\kern-0.02em\bbl@allowhyphens j}
2119 \DeclareTextCommand{\IJ}{OT1}{%
2120 I\kern-0.02em\bbl@allowhyphens J}
2121 \DeclareTextCommand{\ij}{T1}{\char188}
2122 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2123 \ProvideTextCommandDefault{\ij}{%
2124   \UseTextSymbol{OT1}{\ij}}
2125 \ProvideTextCommandDefault{\IJ}{%
2126   \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).

```
2127 \def\crrtic@{\hrule height0.1ex width0.3em}
2128 \def\crttic@{\hrule height0.1ex width0.33em}
2129 \def\ddj@{%
2130   \setbox0\hbox{d}\dimen@=\ht0
2131   \advance\dimen@lex
2132   \dimen@.45\dimen@
2133   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2134   \advance\dimen@ii.5ex
2135   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2136 \def\DDJ@{%
2137   \setbox0\hbox{D}\dimen@=.55\ht0
2138   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2139   \advance\dimen@ii.15ex % correction for the dash position
2140   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2141   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2142   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2143 %
2144 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2145 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2146 \ProvideTextCommandDefault{\dj}{%
2147   \UseTextSymbol{OT1}{\dj}}
2148 \ProvideTextCommandDefault{\DJ}{%
2149   \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.

```
2150 \DeclareTextCommand{\SS}{OT1}{SS}
2151 \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.

```
2152 \ProvideTextCommandDefault{\glq}{%
2153   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2154 \ProvideTextCommand{\grq}{T1}{%
2155   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2156 \ProvideTextCommand{\grq}{TU}{%
2157   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2158 \ProvideTextCommand{\grq}{OT1}{%
2159   \save@sf@q{\kern-.0125em
2160     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%

```

```

2161 \kern.07em\relax}}
2162 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2163 \ProvideTextCommandDefault{\glqq}{%
2164 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2165 \ProvideTextCommand{\grqq}{T1}{%
2166 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2167 \ProvideTextCommand{\grqq}{TU}{%
2168 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2169 \ProvideTextCommand{\grqq}{0T1}{%
2170 \save@sf@q{\kern-.07em
2171 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2172 \kern.07em\relax}}
2173 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2174 \ProvideTextCommandDefault{\flq}{%
2175 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2176 \ProvideTextCommandDefault{\frq}{%
2177 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```

2178 \ProvideTextCommandDefault{\flqq}{%
2179 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2180 \ProvideTextCommandDefault{\frqq}{%
2181 \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).

```

2182 \def\umlauthigh{%
2183 \def\bbl@umlauta##1{\leavevmode\bgroup%
2184 \accent\csname\f@encoding dqpos\endcsname
2185 ##1\bbl@allowhyphens\egroup}%
2186 \let\bbl@umlaute\bbl@umlauta}
2187 \def\umlautlow{%
2188 \def\bbl@umlauta{\protect\lower@umlaut}}
2189 \def\umlautelow{%
2190 \def\bbl@umlaute{\protect\lower@umlaut}}
2191 \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.

```
2192 \expandafter\ifx\csname U@D\endcsname\relax
2193   \csname newdimen\endcsname\U@D
2194 \fi
```

The following code fools T_EX'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.

```
2195 \def\lower@umlaut#1{%
2196   \leavevmode\bgroup
2197     \U@D lex%
2198     {\setbox\z@\hbox{%
2199       \char\csname f@encoding dqpos\endcsname}%
2200       \dimen@ -.45ex\advance\dimen@\ht\z@
2201       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2202     \accent\csname f@encoding dqpos\endcsname
2203     \fontdimen5\font\U@D #1%
2204   \egroup}
```

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

```
2205 \AtBeginDocument{%
2206   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2207   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2208   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2209   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
2210   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2211   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2212   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2213   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2214   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2215   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2216   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}}
```

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

```
2217 \ifx\l@english\@undefined
2218   \chardef\l@english\z@
2219 \fi
2220 % The following is used to cancel rules in ini files (see Amharic).
2221 \ifx\l@unhyphenated\@undefined
2222   \newlanguage\l@unhyphenated
2223 \fi
```

4.16. Layout

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

```
2224 \bbl@trace{Bidi layout}
2225 \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.

```
2226 \bbl@trace{Input engine specific macros}
2227 \ifcase\bbl@engine
2228   \input txtbabel.def
2229 \or
2230   \input luababel.def
2231 \or
2232   \input xebabel.def
2233 \fi
2234 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2235 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2236 \ifx\babelposthyphenation\undefined
2237   \let\babelposthyphenation\babelprehyphenation
2238   \let\babelpatterns\babelprehyphenation
2239   \let\babelcharproperty\babelprehyphenation
2240 \fi
2241 </package | core>
```

4.18. Creating and modifying languages

Continue with $\LaTeX$ only.

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

```
2242 < *package>
2243 \bbl@trace{Creating languages and reading ini files}
2244 \let\bbl@extend@ini@gobble
2245 \newcommand\babelprovide[2][]{%
2246   \let\bbl@save@langname\language@name
2247   \edef\bbl@save@localeid{\the\localeid}%
2248   \global\let\bbl@afterload@empty
2249   % Set name and locale id
2250   \edef\language@name{#2}%
2251   \bbl@id@assign
2252   % Initialize keys
2253   \bbl@vforeach{captions,date,import,main,script,language,%
2254     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2255     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2256     Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2257     @import}%
2258     {\bbl@csarg\let{KVP@##1}\@nnil}%
2259   \global\let\bbl@release@transforms@empty
2260   \global\let\bbl@release@casing@empty
2261   \let\bbl@calendars@empty
2262   \global\let\bbl@inidata@empty
2263   \global\let\bbl@extend@ini@gobble
2264   \global\let\bbl@included@inis@empty
2265   \gdef\bbl@key@list{;}%
2266   \bbl@ifunset{bbl@passto@#2}%
2267     {\def\bbl@tempa{#1}%
2268      {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}%
2269      \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2270        \in@/{#1}% With /, (re)sets a value in the ini
2271        \ifin@
2272          \bbl@renewinikey##1\@@{##2}%
2273        \else
2274          \bbl@csarg\ifx{KVP@##1}\@nnil\else
2275            \bbl@error{unknown-provide-key}{##1}{}}{}%
2276        \fi
2277        \bbl@csarg\def{KVP@##1}{##2}%
```

```

2278 \fi}%
2279 \chardef\bb@howloaded=% 0:none; 1:ldf without ini; 2:ini
2280 \bb@ifunset{date#2}\z@{\bb@ifunset{bb@llevel@#2}\@ne\tw}%
2281 % == init ==
2282 \ifx\bb@screset\@undefined
2283 \bb@ldfinit
2284 \fi
2285 % ==
2286 % If there is no import (last wins), use @import (internal, there
2287 % must be just one). To consider any order (because
2288 % \PassOptionsToLocale).
2289 \ifx\bb@KVP@import\@nnil
2290 \let\bb@KVP@import\bb@KVP@@import
2291 \fi
2292 % == date (as option) ==
2293 % \ifx\bb@KVP@date\@nnil\else
2294 % \fi
2295 % ==
2296 \let\bb@lbbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2297 \ifcase\bb@howloaded
2298 \let\bb@lbbkflag\@empty % new
2299 \else
2300 \ifx\bb@KVP@hyphenrules\@nnil\else
2301 \let\bb@lbbkflag\@empty
2302 \fi
2303 \ifx\bb@KVP@import\@nnil\else
2304 \let\bb@lbbkflag\@empty
2305 \fi
2306 \fi
2307 % == import, captions ==
2308 \ifx\bb@KVP@import\@nnil\else
2309 \bb@exp{\bb@ifblank{\bb@KVP@import}}%
2310 {\ifx\bb@initoload\relax
2311 \begingroup
2312 \def\BabelBeforeIni##1##2{\gdef\bb@KVP@import{##1}\endinput}%
2313 \bb@input@texini{##2}%
2314 \endgroup
2315 \else
2316 \xdef\bb@KVP@import{\bb@initoload}%
2317 \fi}%
2318 {}%
2319 \let\bb@KVP@date\@empty
2320 \fi
2321 \let\bb@KVP@captions@@\bb@KVP@captions
2322 \ifx\bb@KVP@captions\@nnil
2323 \let\bb@KVP@captions\bb@KVP@import
2324 \fi
2325 % ==
2326 \ifx\bb@KVP@transforms\@nnil\else
2327 \bb@replace\bb@KVP@transforms{ },}%
2328 \fi
2329 % ==
2330 \ifx\bb@KVP@mapdot\@nnil\else
2331 \def\bb@tempa{\@empty}%
2332 \ifx\bb@KVP@mapdot\bb@tempa\else
2333 \bb@exp{\gdef<\bb@map@@.@@<\language>{\bb@KVP@mapdot}}}%
2334 \fi
2335 \fi
2336 % Load ini
2337 % -----
2338 \ifcase\bb@howloaded
2339 \bb@provide@new{#2}%
2340 \else

```

```

2341 \bbl@ifblank{#1}%
2342 {}% With \bbl@load@basic below
2343 {\bbl@provide@renew{#2}}%
2344 \fi
2345 % Post tasks
2346 % -----
2347 % == subsequent calls after the first provide for a locale ==
2348 \ifx\bbl@inidata\@empty\else
2349 \bbl@extend@ini{#2}%
2350 \fi
2351 % == ensure captions ==
2352 \ifx\bbl@KVP@captions\@nnil\else
2353 \bbl@ifunset{bbl@extracaps@#2}%
2354 {\bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}}%
2355 {\bbl@exp{\bbl@babelensure[exclude=\\today,
2356 include=\[bbl@extracaps@#2]]{#2}}}%
2357 \let\bbl@tempc\languagename
2358 \bbl@replace\bbl@tempc{-}{@}%
2359 \bbl@ifunset{bbl@ensure@\bbl@tempc}%
2360 {\bbl@exp{%
2361 \\DeclareRobustCommand\<bbl@ensure@\bbl@tempc>[1]{%
2362 \\foreignlanguage{\languagename}%
2363 {###1}}}%
2364 }%
2365 \bbl@exp{%
2366 \\bbl@tglobal\<bbl@ensure@\bbl@tempc>%
2367 \\bbl@tglobal\<bbl@ensure@\bbl@tempc\space>%
2368 \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.

```

2369 \bbl@load@basic{#2}%
2370 % == script, language ==
2371 % Override the values from ini or defines them
2372 \ifx\bbl@KVP@script\@nnil\else
2373 \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2374 \fi
2375 \ifx\bbl@KVP@language\@nnil\else
2376 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2377 \fi
2378 \ifcase\bbl@engine\or
2379 \bbl@ifunset{bbl@chrng@\languagename}{}%
2380 {\directlua{
2381 Babel.set_chrngs_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2382 \fi
2383 % == Line breaking: intraspace, intrapenalty ==
2384 % For CJK, East Asian, Southeast Asian, if interspace in ini
2385 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2386 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2387 \fi
2388 \bbl@provide@intraspace
2389 % == Line breaking: justification ==
2390 \ifx\bbl@KVP@justification\@nnil\else
2391 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2392 \fi
2393 \ifx\bbl@KVP@linebreaking\@nnil\else
2394 \bbl@xin@{,\bbl@KVP@linebreaking,}%
2395 {,elongated,kashida,cjk,padding,unhyphenated,}%
2396 \ifin@
2397 \bbl@csarg\xdef
2398 {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2399 \fi

```

```

2400 \fi
2401 \bbl@xin@{/e}/{/bbl@cl{lnbrk}}%
2402 \ifin@else\bbl@xin@{/k}/{/bbl@cl{lnbrk}}\fi
2403 \ifin@bbl@arabicjust\fi
2404 \bbl@xin@{/p}/{/bbl@cl{lnbrk}}%
2405 \ifin@AtBeginDocument{@nameuse{bbl@tibetanjust}}\fi
2406 % == Line breaking: hyphenate.other.(locale|script) ==
2407 \ifx\bbl@lbfkflag\empty
2408   \bbl@ifunset{bbl@hyotl@language\language}{}%
2409   {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
2410     \bbl@startcommands*{language}{}%
2411     \bbl@csarg\bbl@foreach{hyotl@language}{%
2412       \ifcase\bbl@engine
2413         \ifnum##1<257
2414           \SetHyphenMap{\BabelLower{##1}{##1}}%
2415         \fi
2416       \else
2417         \SetHyphenMap{\BabelLower{##1}{##1}}%
2418       \fi}%
2419   \bbl@endcommands}%
2420 \bbl@ifunset{bbl@hyots@language\language}{}%
2421 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2422   \bbl@csarg\bbl@foreach{hyots@language}{%
2423     \ifcase\bbl@engine
2424       \ifnum##1<257
2425         \global\lccode##1=##1\relax
2426       \fi
2427     \else
2428       \global\lccode##1=##1\relax
2429     \fi}}%
2430 \fi
2431 % == Counters: maparabic ==
2432 % Native digits, if provided in ini (TeX level, xe and lua)
2433 \ifcase\bbl@engine\else
2434   \bbl@ifunset{bbl@dgnat@language\language}{}%
2435   {\expandafter\ifx\csname bbl@dgnat@language\endcsname\empty\else
2436     \expandafter\expandafter\expandafter
2437     \bbl@setdigits\csname bbl@dgnat@language\endcsname
2438     \ifx\bbl@KVP@maparabic\@nnil\else
2439       \ifx\bbl@latinarabic\@undefined
2440         \expandafter\let\expandafter\@arabic
2441         \csname bbl@counter@language\endcsname
2442       \else % i.e., if layout=counters, which redefines \@arabic
2443         \expandafter\let\expandafter\bbl@latinarabic
2444         \csname bbl@counter@language\endcsname
2445       \fi
2446     \fi
2447   \fi}%
2448 \fi
2449 % == Counters: mapdigits ==
2450 % > luababel.def
2451 % == Counters: alph, Alph ==
2452 \ifx\bbl@KVP@alph\@nnil\else
2453   \bbl@exp{%
2454     \\bbl@add<bbl@preextras@language>{%
2455       \\babel@save\\@alph
2456       \let\\@alph<bbl@cntr@bbl@KVP@alph @language>}}%
2457 \fi
2458 \ifx\bbl@KVP@Alph\@nnil\else
2459   \bbl@exp{%
2460     \\bbl@add<bbl@preextras@language>{%
2461       \\babel@save\\@Alph
2462       \let\\@Alph<bbl@cntr@bbl@KVP@Alph @language>}}%

```

```

2463 \fi
2464 % == Counters: mapdot ==
2465 \ifx\bbbl@KVP@mapdot\@nnil\else
2466   \bbbl@foreach\bbbl@list@the{%
2467     \bbbl@ifunset{the##1}{}%
2468     {{\bbbl@ncarg\let\bbbl@tempd{the##1}%
2469       \bbbl@carg\bbbl@sreplace{the##1}{.}{\bbbl@map@lbl{.}}}%
2470       \expandafter\ifx\csname the##1\endcsname\bbbl@tempd\else
2471         \bbbl@exp{\gdef<the##1>{{\the##1}}}%
2472       \fi}}}%
2473   \edef\bbbl@tempb{enumi,enumii,enumiii,enumiv}%
2474   \bbbl@foreach\bbbl@tempb{%
2475     \bbbl@ifunset{label##1}{}%
2476     {{\bbbl@ncarg\let\bbbl@tempd{label##1}%
2477       \bbbl@carg\bbbl@sreplace{label##1}{.}{\bbbl@map@lbl{.}}}%
2478       \expandafter\ifx\csname label##1\endcsname\bbbl@tempd\else
2479         \bbbl@exp{\gdef<label##1>{{\label##1}}}%
2480       \fi}}}%
2481 \fi
2482 % == Casing ==
2483 \bbbl@release@casing
2484 \ifx\bbbl@KVP@casing\@nnil\else
2485   \bbbl@csarg\xdef{casing@{language\name}}%
2486   {\@nameuse{bbbl@casing@{language\name}}\bbbl@maybextx\bbbl@KVP@casing}%
2487 \fi
2488 % == Calendars ==
2489 \ifx\bbbl@KVP@calendar\@nnil
2490   \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}}%
2491 \fi
2492 \def\bbbl@tempe##1 ##2\@@{% Get first calendar
2493   \def\bbbl@tempa{##1}}%
2494   \bbbl@exp{\bbbl@tempe\bbbl@KVP@calendar\space\@@}%
2495 \def\bbbl@tempe##1.##2.##3\@@{%
2496   \def\bbbl@tempc{##1}%
2497   \def\bbbl@tempb{##2}}%
2498 \expandafter\bbbl@tempe\bbbl@tempa..\@@
2499 \bbbl@csarg\edef{calpr@{language\name}}{%
2500   \ifx\bbbl@tempc@empty\else
2501     calendar=\bbbl@tempc
2502   \fi
2503   \ifx\bbbl@tempb@empty\else
2504     ,variant=\bbbl@tempb
2505   \fi}%
2506 % == engine specific extensions ==
2507 % Defined in XXXbabel.def
2508 \bbbl@provide@extra{#2}%
2509 % == require.babel in ini ==
2510 % To load or reload the babel-*.tex, if require.babel in ini
2511 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2512   \bbbl@ifunset{bbbl@rqtex@{language\name}}{%
2513     {\expandafter\ifx\csname bbbl@rqtex@{language\name}\endcsname\@empty\else
2514       \let\BabelBeforeIni\@gobbletwo
2515       \chardef\atcatcode=\catcode`\@
2516       \catcode`\@=11\relax
2517       \def\CurrentOption{#2}%
2518       \bbbl@input@texini{\bbbl@cs{rqtex@{language\name}}}%
2519       \catcode`\@=\atcatcode
2520       \let\atcatcode\relax
2521       \global\bbbl@csarg\let{rqtex@{language\name}}\relax
2522     \fi}%
2523   \bbbl@foreach\bbbl@calendars{%
2524     \bbbl@ifunset{bbbl@ca@##1}{%
2525       \chardef\atcatcode=\catcode`\@

```

```

2526      \catcode`\@=11\relax
2527      \InputIfFileExists{babel-ca-##1.tex}{}{}%
2528      \catcode`\@=\atcatcode
2529      \let\atcatcode\relax}%
2530  {}}%
2531  \fi
2532  % == frenchspacing ==
2533  \ifcase\bbl@howloaded\in@true\else\in@false\fi
2534  \ifin@ \else \bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2535  \ifin@
2536    \bbl@extras@wrap{\bbl@pre@fs}%
2537    {\bbl@pre@fs}%
2538    {\bbl@post@fs}%
2539  \fi
2540  % == transforms ==
2541  % > luababel.def
2542  \def\CurrentOption{#2}%
2543  \@nameuse{\bbl@icsave@#2}%
2544  % == main ==
2545  \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2546    \let\languagename\bbl@savelangname
2547    \chardef\localeid\bbl@savelocaleid\relax
2548  \fi
2549  % == ==
2550  \ifx\ldf@finish\@onlypreamble\else
2551    \bbl@afterload
2552  \fi
2553  % == hyphenrules (apply if current) ==
2554  \ifx\bbl@KVP@hyphenrules\@nnil\else
2555    \ifnum\bbl@savelocaleid=\localeid
2556      \language\@nameuse{l@\languagename}%
2557    \fi
2558  \fi}

```

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

```

2559 \def\bbl@provide@new#1{%
2560   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2561   \@namedef{extras#1}{}%
2562   \@namedef{noextras#1}{}%
2563   \bbl@startcommands*{#1}{captions}%
2564   \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2565     \def\bbl@tempb##1{% elt for \bbl@captionslist
2566       \ifx##1\@nnil\else
2567         \bbl@exp{%
2568           \\SetString\\##1{%
2569             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2570         \expandafter\bbl@tempb
2571       \fi}%
2572     \expandafter\bbl@tempb\bbl@captionslist\@nnil
2573   \else
2574     \ifx\bbl@initoload\relax
2575       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2576     \else
2577       \bbl@read@ini{\bbl@initoload}2% % Same
2578     \fi
2579   \fi
2580   \StartBabelCommands*{#1}{date}%
2581   \ifx\bbl@KVP@date\@nnil
2582     \bbl@exp{%
2583       \\SetString\\today{\bbl@nocaption{today}{#1today}}}%
2584   \else
2585     \bbl@savetoday

```

```

2586     \bbl@savestate
2587     \fi
2588 \bbl@endcommands
2589 \bbl@load@basic{#1}%
2590 % == hyphenmins == (only if new)
2591 \bbl@exp{%
2592     \gdef\<#1hyphenmins>{%
2593         {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2594         {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2595 % == hyphenrules (also in renew) ==
2596 \bbl@provide@hyphens{#1}%
2597 % == main ==
2598 \ifx\bbl@KVP@main\@nnil\else
2599     \expandafter\main@language\expandafter{#1}%
2600 \fi}
2601 %
2602 \def\bbl@provide@renew#1{%
2603     \ifx\bbl@KVP@captions\@nnil\else
2604         \StartBabelCommands*{#1}{captions}%
2605         \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2606         \EndBabelCommands
2607     \fi
2608     \ifx\bbl@KVP@date\@nnil\else
2609         \StartBabelCommands*{#1}{date}%
2610         \bbl@savestate
2611         \bbl@savestate
2612         \EndBabelCommands
2613     \fi
2614     % == hyphenrules (also in new) ==
2615     \ifx\bbl@lbkflag\@empty
2616         \bbl@provide@hyphens{#1}%
2617     \fi
2618     % == main ==
2619     \ifx\bbl@KVP@main\@nnil\else
2620         \expandafter\main@language\expandafter{#1}%
2621     \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.

```

2622 \def\bbl@load@basic#1{%
2623     \ifcase\bbl@howloaded\or\or
2624         \ifcase\csname bbl@llevel@\language\endcsname
2625             \bbl@csarg\let{lname@\language}\relax
2626         \fi
2627     \fi
2628     \bbl@ifunset{\bbl@lname@#1}%
2629     {\def\BabelBeforeIni##1##2{%
2630         \begingroup
2631             \let\bbl@ini@captions@aux\@gobbletwo
2632             \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2633             \bbl@read@ini{##1}1%
2634             \ifx\bbl@initoload\relax\endinput\fi
2635         \endgroup}%
2636     \begingroup    % boxed, to avoid extra spaces:
2637         \ifx\bbl@initoload\relax
2638             \bbl@input@texini{#1}%
2639         \else
2640             \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2641         \fi
2642     \endgroup}%
2643     {}}

```

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).

```

2644 \def\bbl@load@info#1{%
2645   \def\BabelBeforeIni##1##2{%
2646     \begingroup
2647       \bbl@read@ini{##1}0%
2648       \endinput           % babel- .tex may contain onlypreamble's
2649     \endgroup}%          boxed, to avoid extra spaces:
2650   {\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.

```

2651 \def\bbl@provide@hyphens#1{%
2652   \@tempcnta\m@ne % a flag
2653   \ifx\bbl@KVP@hyphenrules\@nnil\else
2654     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2655     \bbl@foreach\bbl@KVP@hyphenrules{%
2656       \ifnum\@tempcnta=\m@ne % if not yet found
2657         \bbl@ifsamestring{##1}{+}%
2658         {\bbl@carg\addlanguage{l@##1}}%
2659         }%
2660       \bbl@ifunset{l@##1}% After a possible +
2661       {}%
2662       {\@tempcnta\@nameuse{l@##1}}%
2663     \fi}%
2664   \ifnum\@tempcnta=\m@ne
2665     \bbl@warning{%
2666       Requested 'hyphenrules' for '\language' not found:\\%
2667       \bbl@KVP@hyphenrules.\\%
2668       Using the default value. Reported}%
2669   \fi
2670 \fi
2671 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2672   \ifx\bbl@KVP@captions@\@nnil
2673     \bbl@ifunset\bbl@hyphr@#1{}% use value in ini, if exists
2674     {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2675      {}%
2676      {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2677      {}% if hyphenrules found:
2678      {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2679   \fi
2680 \fi
2681 \bbl@ifunset{l@#1}%
2682   {\ifnum\@tempcnta=\m@ne
2683     \bbl@carg\adddialect{l@#1}\language
2684   \else
2685     \bbl@carg\adddialect{l@#1}\@tempcnta
2686   \fi}%
2687   {\ifnum\@tempcnta=\m@ne\else
2688     \global\bbl@carg\chardef{l@#1}\@tempcnta
2689   \fi}}
```

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

```

2690 \def\bbl@input@texini#1{%
2691   \bbl@bsphack
2692   \bbl@exp{%
2693     \catcode`\\%=14 \catcode`\\\%=0
2694     \catcode`\\{=1 \catcode`\\\}=2
2695     \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2696     \catcode`\\%=the\catcode`\\%relax
2697     \catcode`\\\%=the\catcode`\\\%relax
```

```

2698      \catcode`\\={\the\catcode`\}\relax
2699      \catcode`\\}= \the\catcode`\}\relax}%
2700      \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.

2701 \def\bbl@iniline#1\bbl@iniline{%
2702   \ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2703 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2704 \def\bbl@iniskip#1\@@{%      if starts with ;
2705 \def\bbl@inistore#1=#2\@@{%   full (default)
2706   \bbl@trim@def\bbl@tempa{#1}%
2707   \bbl@trim\toks@{#2}%
2708   \bbl@ifsamestring{\bbl@tempa}{\include}%
2709   {\bbl@read@subini{\the\toks@}}%
2710   {\bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2711    \ifin@ \else
2712      \bbl@xin@{,identification/include.}%
2713      {,\bbl@section/\bbl@tempa}%
2714      \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2715      \bbl@exp{%
2716        \\g@addto@macro\\bbl@inidata{%
2717          \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2718      \fi}}
2719 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2720   \bbl@trim@def\bbl@tempa{#1}%
2721   \bbl@trim\toks@{#2}%
2722   \bbl@xin@{.identification.}{.\bbl@section.}%
2723   \ifin@
2724     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2725       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2726   \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.

```

2727 \def\bbl@loop@ini#1{%
2728   \loop
2729     \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2730     \endlinechar\m@ne
2731     \read#1 to \bbl@line
2732     \endlinechar`\^^M
2733     \ifx\bbl@line\empty\else
2734       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2735     \fi
2736   \repeat}
2737 %
2738 \def\bbl@read@subini#1{%
2739   \ifx\bbl@readsubstream\undefined
2740     \csname newread\endcsname\bbl@readsubstream
2741   \fi

```

```

2742 \openin\bbl@readsubstream=babel-#1.ini
2743 \ifeof\bbl@readsubstream
2744   \bbl@error{no-ini-file}{#1}{}{}%
2745 \else
2746   {\bbl@loop@ini\bbl@readsubstream}%
2747 \fi
2748 \closein\bbl@readsubstream}
2749 %
2750 \ifx\bbl@readstream\undefined
2751   \csname newread\endcsname\bbl@readstream
2752 \fi
2753 \def\bbl@read@ini#1#2{%
2754   \global\let\bbl@extend@ini@gobble
2755   \openin\bbl@readstream=babel-#1.ini
2756   \ifeof\bbl@readstream
2757     \bbl@error{no-ini-file}{#1}{}{}%
2758   \else
2759     % == Store ini data in \bbl@inidata ==
2760     \catcode`\ =10 \catcode`\"=12
2761     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2762     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2763     \ifnum#2=\m@ne % Just for the info
2764       \edef\language\language{tag \bbl@metalang}%
2765     \fi
2766     \ifnum#2=\m@ne
2767       \bbl@info{Metadata: '\bbl@metalang' requested, '#1' provided.\\%
2768         Fetching the locale name from babel-#1.ini@gobble}%
2769     \else
2770       \bbl@info{Importing
2771         \ifcase#2font and identification \or basic \fi
2772         data for '\language'\\%
2773         from babel-#1.ini. Reported}%
2774     \fi
2775     \ifnum#2<\@ne
2776       \global\let\bbl@inidata\empty
2777       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2778     \fi
2779     \def\bbl@section{identification}%
2780     \bbl@exp{%
2781       \\ \bbl@inistore tag.ini=#1\\ @@
2782       \\ \bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\ @@}%
2783     \bbl@loop@ini\bbl@readstream
2784     % == Process stored data ==
2785     \ifnum#2=\m@ne
2786       \def\bbl@tempa##1 ##2\@@{##1}% Get first name
2787       \def\bbl@elt###1##2###3{%
2788         \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2789         {\edef\language\language{\bbl@tempa##3 \@@}%
2790          \let\locale\language
2791          \bbl@id@assign
2792          \def\bbl@elt####1####2####3{}}%
2793         {}}%
2794       \bbl@inidata
2795     \fi
2796     \bbl@csarg\xdef{\l@ini\language}{#1}%
2797     \bbl@read@ini@aux
2798     % == 'Export' data ==
2799     \bbl@ini@exports{#2}%
2800     \global\bbl@csarg\let{\inidata@\language}\bbl@inidata
2801     \global\let\bbl@inidata\empty
2802     \bbl@xin@{\language,}{\bbl@ini@loaded,}%
2803     \ifin@else
2804       \bbl@exp{\\ \bbl@add@list\\ \bbl@ini@loaded{\language}}%

```

```

2805 \bbl@tglobal\bbl@ini@loaded
2806 \fi
2807 \@ifpackagewith{babel}{licr=unextended}}{%
2808   {\ifcase\bbl@engine\else % Find a better place
2809     \bbl@xin@{\bbl@cl{sbcpr},}{,CyrL,}%
2810     \ifin@
2811       \bbl@once{licr-cryl}{\g@addto@macro\bbl@afterload{\input{cyrLuni.def}}}%
2812     \fi
2813   \fi}%
2814 \fi
2815 \closein\bbl@readstream}
2816 \def\bbl@read@ini@aux{%
2817   \let\bbl@savestrings\@empty
2818   \let\bbl@savetoday\@empty
2819   \let\bbl@savestate\@empty
2820   \def\bbl@elt##1##2##3{%
2821     \def\bbl@section{##1}%
2822     \in@{=date.}{=##1}% Find a better place
2823     \ifin@
2824       \bbl@ifunset{bbl@inikv@##1}%
2825       {\bbl@ini@calendar{##1}}%
2826       {}%
2827     \fi
2828     \bbl@ifunset{bbl@inikv@##1}}{%
2829     {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2830   \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.

```

2831 \def\bbl@extend@ini@aux#1{%
2832   \bbl@startcommands*{#1}{captions}%
2833   % Activate captions/... and modify exports
2834   \bbl@csarg\def{inikv@captions.licr}##1##2{%
2835     \setlocalecaption{#1}{##1}{##2}}%
2836   \def\bbl@inikv@captions##1##2{%
2837     \bbl@ini@captions@aux{##1}{##2}}%
2838   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2839   \def\bbl@exportkey##1##2##3{%
2840     \bbl@ifunset{bbl@kv@##2}}{%
2841     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2842       \bbl@exp{\global\let<bbl@##1\language>\<bbl@kv@##2>}}%
2843     \fi}}%
2844   % As with \bbl@read@ini, but with some changes
2845   \bbl@read@ini@aux
2846   \bbl@ini@exports\tw@
2847   % Update inidata@lang by pretending the ini is read.
2848   \def\bbl@elt##1##2##3{%
2849     \def\bbl@section{##1}%
2850     \bbl@iniline##2=##3\bbl@iniline}%
2851     \csname bbl@inidata@##1\endcsname
2852     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2853   \StartBabelCommands*{#1}{date}% And from the import stuff
2854   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2855   \bbl@savetoday
2856   \bbl@savestate
2857   \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2858 \def\bbl@ini@calendar#1{%
2859   \lowercase{\def\bbl@tempa{=#1=}}%
2860   \bbl@replace\bbl@tempa{=date.gregorian.}}{%
2861   \bbl@replace\bbl@tempa{=date.}}{%
2862   \in@{.licr=}{#1=}%
2863   \ifin@

```

```

2864 \ifcase\bbl@engine
2865   \bbl@replace\bbl@tempa{.licr={}}}%
2866 \else
2867   \let\bbl@tempa\relax
2868 \fi
2869 \fi
2870 \ifx\bbl@tempa\relax\else
2871   \bbl@replace\bbl@tempa{=}{}}%
2872 \ifx\bbl@tempa\empty\else
2873   \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2874 \fi
2875 \bbl@exp{%
2876   \def\<bbl@inikv@#1>####1####2{%
2877     \<bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2878 \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).

```

2879 \def\bbl@renewinikey#1/#2\@#3{%
2880   \global\let\bbl@extend@ini\bbl@extend@ini@aux
2881   \edef\bbl@tempa{\zap@space #1 \empty}%   section
2882   \edef\bbl@tempb{\zap@space #2 \empty}%   key
2883   \bbl@trim\toks@{#3}%                     value
2884   \bbl@exp{%
2885     \edef\<bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2886     \<\gaddto@macro\<\bbl@inidata{%
2887       \<\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.

```

2888 \def\bbl@exportkey#1#2#3{%
2889   \bbl@ifunset{bbl@kv@#2}%
2890   {\bbl@csarg\gdef{#1@ \language name}{#3}}%
2891   {\expandafter\ifx\csname bbl@kv@#2\endcsname\empty
2892     \bbl@csarg\gdef{#1@ \language name}{#3}%
2893   \else
2894     \bbl@exp{\global\let\<bbl@#1@ \language name>\<bbl@kv@#2>}%
2895   \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.

```

2896 \def\bbl@iniwarning#1{%
2897   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2898   {\bbl@warning{%
2899     From babel-\bbl@cs{lini@ \language name}.ini:\%
2900     \bbl@cs{kv@identification.warning#1}\%
2901     Reported}}}
2902 %
2903 \let\bbl@release@transforms\empty
2904 \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).

```

2905 \def\bbl@ini@exports#1{%
2906   % Identification always exported
2907   \bbl@iniwarning{}%
2908   \ifcase\bbl@engine
2909     \bbl@iniwarning{.pdflatex}%
2910   \or
2911     \bbl@iniwarning{.lualatex}%
2912   \or
2913     \bbl@iniwarning{.xelatex}%
2914   \fi%
2915   \bbl@exportkey{lllevel}{identification.load.level}{}%
2916   \bbl@exportkey{elname}{identification.name.english}{}%
2917   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2918     {\csname bbl@elname@language\endcsname}}%
2919   \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2920   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2921   \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2922   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2923   \bbl@exportkey{esname}{identification.script.name}{}%
2924   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2925     {\csname bbl@esname@language\endcsname}}%
2926   \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2927   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2928   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2929   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2930   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2931   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2932   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2933   % Also maps bcp47 -> language
2934   \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{\language}%
2935   \ifcase\bbl@engine\or
2936     \directlua{%
2937       Babel.locale_props[\the\bbl@cs{id@language}].script
2938       = '\bbl@cl{sbcpl}'}%
2939   \fi
2940   % Conditional
2941   \ifnum#1>\z@ % -1 or 0 = only info, 1 = basic, 2 = (re)new
2942     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2943     \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
2944     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2945     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2946     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2947     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2948     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2949     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2950     \bbl@exportkey{intsp}{typography.intraspace}{}%
2951     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2952     \bbl@exportkey{chrng}{characters.ranges}{}%
2953     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2954     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2955     \ifnum#1=\tw@ % only (re)new
2956       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2957       \bbl@tglobal\bbl@savetoday
2958       \bbl@tglobal\bbl@savestate
2959       \bbl@savestrings
2960     \fi
2961   \fi}

```

4.20. Processing keys in ini

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

```
2962 \def\bbl@inikv#1#2{%      key=value
2963   \toks@{#2}%              This hides #'s from ini values
2964   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
2965 \let\bbl@inikv@identification\bbl@inikv
2966 \let\bbl@inikv@date\bbl@inikv
2967 \let\bbl@inikv@typography\bbl@inikv
2968 \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.

```
2969 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\empty x-\fi}
2970 \def\bbl@inikv@characters#1#2{%
2971   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2972   {\bbl@exp{%
2973     \\g@addto@macro\\bbl@release@casing{%
2974       \\bbl@casemapping}{\language}\unexpanded{#2}}}%
2975   {\in@{$casing.}{$#1}% e.g., casing.Uv = uV
2976   \ifin@
2977     \lowercase{\def\bbl@tempb{#1}}%
2978     \bbl@replace\bbl@tempb{casing.}{}%
2979     \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2980       \\bbl@casemapping
2981       {\\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2982   \else
2983     \bbl@inikv{#1}{#2}%
2984   \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’.

```
2985 \def\bbl@inikv@counters#1#2{%
2986   \bbl@ifsamestring{#1}{digits}%
2987   {\bbl@error{digits-is-reserved}{}}{}%
2988   {}%
2989   \def\bbl@tempc{#1}%
2990   \bbl@trim@def{\bbl@tempb*}{#2}%
2991   \in@{.1$}{#1$}%
2992   \ifin@
2993     \bbl@replace\bbl@tempc{.1}{}%
2994     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
2995       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2996   \fi
2997   \in@{.F.}{#1}%
2998   \ifin@\else\in@{.S.}{#1}\fi
2999   \ifin@
3000     \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3001   \else
3002     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3003     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3004     \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3005   \fi
3006   \in@{.D$}{#1$}%
3007   \ifin@\else\in@{.C$}{#1$}\fi
3008   \ifin@
3009     \bbl@replace\bbl@tempc{.D}{}%
3010     \bbl@replace\bbl@tempc{.C}{}%
3011     \bbl@exp{\gdef\<bbl@cntr@\bbl@tempc @\language>###1{%
3012       \\expandafter\\bbl@zhnumeral\\expandafter{\number###1}{\bbl@tempc}}}%

```

3013 \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.

```
3014 \ifcase\bbl@engine
3015 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3016 \bbl@ini@captions@aux{#1}{#2}}
3017 \else
3018 \def\bbl@inikv@captions#1#2{%
3019 \bbl@ini@captions@aux{#1}{#2}}
3020 \fi
```

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

```
3021 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3022 \bbl@replace\bbl@tempa{.template}{}%
3023 \def\bbl@toreplace{#1}{}%
3024 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3025 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3026 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3027 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname{}}%
3028 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3029 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3030 \ifin@
3031 \nameuse{bbl@patch\bbl@tempa}%
3032 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3033 \fi
3034 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3035 \ifin@
3036 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3037 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3038 \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\\language}%
3039 {\fnum@\bbl@tempa}}%
3040 {\\\@nameuse{bbl@\bbl@tempa fmt@\\\language}}}%
3041 \fi}
3042 %
3043 \def\bbl@ini@captions@aux#1#2{%
3044 \bbl@trim@def\bbl@tempa{#1}%
3045 \bbl@xin@{.template}{\bbl@tempa}%
3046 \ifin@
3047 \bbl@ini@captions@template{#2}\language
3048 \else
3049 \bbl@ifblank{#2}%
3050 {\bbl@exp{%
3051 \toks@{\\\bbl@nocaption{\bbl@tempa name}{\language\bbl@tempa name}}}%
3052 {\bbl@trim\toks@{#2}}}%
3053 \bbl@exp{%
3054 \\\bbl@add\\bbl@savestrings{%
3055 \\\SetString\<\bbl@tempa name>\the\toks@}}%
3056 \toks@\expandafter{\bbl@captionslist}%
3057 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3058 \ifin@else
3059 \bbl@exp{%
3060 \\\bbl@add\<bbl@extracaps@\language>\<\bbl@tempa name>}%
3061 \\\bbl@to\global\<bbl@extracaps@\language>}%
3062 \fi
3063 \fi}
```

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

```
3064 \def\bbl@list@the{%
3065 part,chapter,section,subsection,subsubsection,paragraph,%
3066 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3067 table,page,footnote,mpfootnote,mpfn}
3068 %
```

```

3069 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3070 \bbl@ifunset\bbl@map@#1@\language}%
3071 {\@nameuse{#1}}%
3072 {\@nameuse\bbl@map@#1@\language}}
3073 %
3074 \def\bbl@map@lbl#1{% #1:a sign, eg, .
3075 \ifincsname#1\else
3076 \bbl@ifunset\bbl@map@#1@\language}%
3077 {#1}%
3078 {\@nameuse\bbl@map@#1@\language}}%
3079 \fi}
3080 %
3081 \def\bbl@inikv@labels#1#2{%
3082 \in@{.map}{#1}%
3083 \ifin@
3084 \in@{,dot.map,}{, #1,}%
3085 \ifin@
3086 \global\@namedef\bbl@map@.@@\language}{#2}%
3087 \fi
3088 \ifx\bbl@KVP@labels\@nnil\else
3089 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3090 \ifin@
3091 \def\bbl@tempc{#1}%
3092 \bbl@replace\bbl@tempc{.map}{}%
3093 \in@{, #2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3094 \bbl@exp{%
3095 \gdef\<bbl@map@\bbl@tempc @\language>%
3096 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3097 \bbl@foreach\bbl@list@the{%
3098 \bbl@ifunset{the##1}{}%
3099 {\bbl@ncarg\let\bbl@tempd{the##1}%
3100 \bbl@exp{%
3101 \\bbl@sreplace\<the##1>%
3102 {\<\bbl@tempc>{##1}}%
3103 {\bbl@map@cnt{\bbl@tempc}{##1}}%
3104 \\bbl@sreplace\<the##1>%
3105 {\<\@empty @\bbl@tempc>\<c@##1>%
3106 {\bbl@map@cnt{\bbl@tempc}{##1}}%
3107 \\bbl@sreplace\<the##1>%
3108 {\bbl@tempc\\endcsname\<c@##1>%
3109 {\bbl@map@cnt{\bbl@tempc}{##1}}}%
3110 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3111 \bbl@exp{\gdef\<the##1>{\[the##1]}}%
3112 \fi}}%
3113 \fi
3114 \fi
3115 %
3116 \else
3117 % The following code is still under study. You can test it and make
3118 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3119 % language dependent.
3120 \in@{enumerate.}{#1}%
3121 \ifin@
3122 \def\bbl@tempa{#1}%
3123 \bbl@replace\bbl@tempa{enumerate.}{}%
3124 \def\bbl@toreplace{#2}%
3125 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3126 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3127 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3128 \toks@{\expandafter\bbl@toreplace}%
3129 \bbl@exp{%
3130 \\bbl@add\<extras\language>%
3131 \\babel@save\<labelenum\romannumeral\bbl@tempa>%

```

```

3132         \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3133     \\bbl@tglobal<extras\language>}%
3134     \fi
3135 \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.

```

3136 \def\bbl@chapttype{chapter}
3137 \ifx\@makechapterhead\undefined
3138     \let\bbl@patchchapter\relax
3139 \else\ifx\thechapter\undefined
3140     \let\bbl@patchchapter\relax
3141 \else\ifx\ps@headings\undefined
3142     \let\bbl@patchchapter\relax
3143 \else
3144     \def\bbl@patchchapter{%
3145         \global\let\bbl@patchchapter\relax
3146         \gdef\bbl@chfmt{%
3147             \bbl@ifunset{bbl@\bbl@chapttype fmt@\language}%
3148                 {\@chapapp\space\thechapter}%
3149                 {\@nameuse{bbl@\bbl@chapttype fmt@\language}}}%
3150         \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3151         \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3152         \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3153         \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3154         \bbl@tglobal\appendix
3155         \bbl@tglobal\ps@headings
3156         \bbl@tglobal\chaptermark
3157         \bbl@tglobal\@makechapterhead}
3158     \let\bbl@patchappendix\bbl@patchchapter
3159 \fi\fi\fi
3160 \ifx\@part\undefined
3161     \let\bbl@patchpart\relax
3162 \else
3163     \def\bbl@patchpart{%
3164         \global\let\bbl@patchpart\relax
3165         \gdef\bbl@partformat{%
3166             \bbl@ifunset{bbl@partfmt@\language}%
3167                 {\partname\nobreakspace\thepart}%
3168                 {\@nameuse{bbl@partfmt@\language}}}%
3169         \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3170         \bbl@tglobal\@part}
3171 \fi

```

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

```

3172 \let\bbl@calendar\@empty
3173 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3174 \def\bbl@localedate#1#2#3#4{%
3175     \begingroup
3176         \edef\bbl@they{#2}%
3177         \edef\bbl@them{#3}%
3178         \edef\bbl@thed{#4}%
3179         \edef\bbl@tempe{%
3180             \bbl@ifunset{bbl@calpr@\language}{\bbl@clcalpr}},%
3181             #1}%
3182         \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3183         \bbl@replace\bbl@tempe{ }{}%
3184         \bbl@replace\bbl@tempe{convert}{convert=}%
3185         \let\bbl@ld@calendar\@empty
3186         \let\bbl@ld@variant\@empty
3187         \let\bbl@ld@convert\relax

```

```

3188 \def\bbl@tempb##1=##2\@@{\@namedef\bbl@ld@##1}{##2}}%
3189 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3190 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3191 \ifx\bbl@ld@calendar\@empty\else
3192 \ifx\bbl@ld@convert\relax\else
3193 \ifx\bbl@ld@convert\@empty
3194 \let\bbl@ld@convert\bbl@ld@calendar
3195 \else
3196 \edef\bbl@ld@convert{\expandafter\@gobble\bbl@ld@convert}%
3197 \fi
3198 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3199 {\bbl@ld@convert}\bbl@they\bbl@them\bbl@thed
3200 \fi
3201 \fi
3202 \@nameuse\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3203 \edef\bbl@calendar{% Used in \month..., too
3204 \bbl@ld@calendar
3205 \ifx\bbl@ld@variant\@empty\else
3206 .\bbl@ld@variant
3207 \fi}%
3208 \bbl@cased
3209 {\@nameuse\bbl@date@\language @\bbl@calendar}%
3210 \bbl@they\bbl@them\bbl@thed}%
3211 \endgroup}
3212 %
3213 \def\bbl@printdate#1{%
3214 \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}]%
3215 \def\bbl@printdate@i#1[#2]#3#4#5{%
3216 \bbl@usedategrouptue
3217 \def\bbl@tempc{#1}%
3218 \bbl@replace\bbl@tempc{-}{\@@}%
3219 \@nameuse\bbl@ensure@\bbl@tempc}{\localedate[#2]{#3}{#4}{#5}}%
3220 %
3221 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3222 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3223 \bbl@trim@def\bbl@tempa{#1.#2}%
3224 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3225 {\bbl@trim@def\bbl@tempa{#3}%
3226 \bbl@trim\toks@{#5}%
3227 \@temptokena\expandafter{\bbl@savedate}%
3228 \bbl@exp{% Reverse order - in ini last wins
3229 \def\\bbl@savedate{%
3230 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3231 \the\@temptokena}}}%
3232 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3233 {\lowercase{\def\bbl@tempb{#6}}}%
3234 \bbl@trim@def\bbl@toreplace{#5}%
3235 \bbl@TG@@date
3236 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3237 \ifx\bbl@savetoday\@empty
3238 \bbl@exp{%
3239 \\AfterBabelCommands{%
3240 \gdef\<\language date>{\protect\<\language date >}%
3241 \gdef\<\language date >{\bbl@printdate{\language}}}%
3242 \def\\bbl@savetoday{%
3243 \\SetString\\today{%
3244 \<\language date>[convert]%
3245 {\the\year}{\the\month}{\the\day}}}%
3246 \fi}%
3247 {}}}}

```

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).

```

3248 \let\bbl@calendar\@empty
3249 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3250   \bbl@xin@{$calendrica:}{$#2}%
3251   \ifin@
3252     \directlua{Babel.calendrica = Babel.calendrica or require("calendrica")}%
3253     \edef\bbl@tempa{ $#2}%
3254     \bbl@replace\bbl@tempa{$calendrica:}{}%
3255     \bbl@replace\bbl@tempa{-}{_}%
3256     \bbl@afterelse
3257     \bbl@exp{\bbl@calendrica#1\\ \@{\bbl@tempa}}%
3258   \else
3259     \bbl@afterfi
3260     \@nameuse{bbl@ca#2}#1\@@
3261   \fi}
3262 \def\bbl@calendrica#1-#2-#3\@@#4#5#6#7{%
3263   \directlua{
3264     local d = Babel.calendrica.new_date(
3265       {year=\number#1, month=\number#2, day=\number#3},
3266       Babel.calendrica.cal_gregorian)
3267     local r = Babel.calendrica.as_date(d, Babel.calendrica.cal_#4)
3268     token.set_macro('\bbl@stripslash#5', r.year)
3269     token.set_macro('\bbl@stripslash#6', r.month)
3270     token.set_macro('\bbl@stripslash#7', r.day) }}
3271 \newcommand\BabelDateSpace{\nobreakspace}
3272 \newcommand\BabelDateDot{.\@}
3273 \newcommand\BabelDated[1]{\{\number#1\}}
3274 \newcommand\BabelDatedd[1]{\{\ifnum#1<10 0\fi\number#1\}}
3275 \newcommand\BabelDateM[1]{\{\number#1\}}
3276 \newcommand\BabelDateMM[1]{\{\ifnum#1<10 0\fi\number#1\}}
3277 \newcommand\BabelDateMMMM[1]{\{%
3278   \csname month\romannumeral#1\bbl@calendar name\endcsname\}}%
3279 \newcommand\BabelDatey[1]{\{\number#1\}}%
3280 \newcommand\BabelDateyy[1]{\{%
3281   \ifnum#1<10 0\number#1 %
3282   \else\ifnum#1<100 \number#1 %
3283   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3284   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3285   \else
3286     \bbl@error{limit-two-digits}{\}\}\}%
3287   \fi\fi\fi\fi}}
3288 \newcommand\BabelDateyyyy[1]{\{\number#1\}}
3289 \newcommand\BabelDateU[1]{\{\number#1\}}%
3290 \def\bbl@replace@finish@iii#1{%
3291   \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3292 \def\bbl@TG@@date{%
3293   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3294   \bbl@replace\bbl@toreplace{[. ]}{\BabelDateDot}}%
3295   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3296   \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[####1|]}%
3297   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3298   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3299   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3300   \bbl@replace\bbl@toreplace{[M|]}{\bbl@datecctr[####2|]}%
3301   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3302   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3303   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3304   \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[####3|]}%
3305   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3306   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}%
3307   \bbl@replace\bbl@toreplace{[U|]}{\bbl@datecctr[####1|]}%
3308   \bbl@replace@finish@iii\bbl@toreplace}

```

```

3309 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3310 \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.

```

3311 \AddToHook{begindocument/before}{%
3312   \let\bbl@normalsf\normalsfcodes
3313   \let\normalsfcodes\relax}
3314 \AtBeginDocument{%
3315   \ifx\bbl@normalsf\empty
3316     \ifnum\sfcodes\@m
3317       \let\normalsfcodes\frenchspacing
3318     \else
3319       \let\normalsfcodes\nonfrenchspacing
3320     \fi
3321   \else
3322     \let\normalsfcodes\bbl@normalsf
3323   \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.

```

3324 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3325 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3326 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3327   #1[#2]{#3}{#4}{#5}}
3328 \begingroup
3329   \catcode`\%=12
3330   \catcode`\&=14
3331   \gdef\bbl@transforms#1#2#3{%&
3332     \directlua{
3333       local str = [=[#2]=]
3334       str = str:gsub('%.%d+%.%d+$', '')
3335       token.set_macro('babeltempa', str)
3336     }&
3337     \def\babeltempc{}&
3338     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3339     \ifin@else
3340       \bbl@xin@{:,\babeltempa,}{,\bbl@KVP@transforms,}&
3341     \fi
3342     \ifin@
3343       \bbl@foreach\bbl@KVP@transforms{%&
3344         \bbl@xin@{:,\babeltempa,}{,##1,}&
3345         \ifin@ & font:font:transform syntax
3346         \directlua{
3347           local t = {}
3348           for m in string.gmatch('##1'..' ':'(.)') do
3349             table.insert(t, m)
3350           end
3351           table.remove(t)
3352           token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3353         }&
3354       \fi}&
3355     \in@{.0$}{#2$}&
3356     \ifin@
3357       \directlua{%& (\attribute) syntax
3358         local str = string.match([[ \bbl@KVP@transforms]],
3359           '%(([^%(-)]^%)[^%)]-\babeltempa')

```

```

3360         if str == nil then
3361             token.set_macro('babeltempb', '')
3362         else
3363             token.set_macro('babeltempb', ',attribute=' .. str)
3364         end
3365     }&%
3366     \toks@{#3}&%
3367     \bbl@exp{&%
3368         \\g@addto@macro\\bbl@release@transforms{&%
3369             \relax &% Closes previous \bbl@transforms@aux
3370             \\bbl@transforms@aux
3371             \\\#1{label=\babeltempa\babeltempb\babeltempc}&%
3372             {\language\the\toks@}}&%
3373     \else
3374         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3375     \fi
3376 \fi}
3377 \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.

```

3378 \def\bbl@provide@lsys#1{%
3379     \bbl@ifunset{bbl@lname@#1}%
3380     {\bbl@load@info{#1}}%
3381     {}%
3382     \bbl@csarg\let{lsys@#1}\@empty
3383     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3384     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3385     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3386     \bbl@ifunset{bbl@lname@#1}{}%
3387     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3388     \ifcase\bbl@engine\or\or
3389     \bbl@ifunset{bbl@prehc@#1}{}%
3390     {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3391     {}%
3392     {\ifx\bbl@xenohyph\undefined
3393         \global\let\bbl@xenohyph\bbl@xenohyph@
3394         \ifx\AtBeginDocument\@notprerr
3395             \expandafter\@secondoftwo % to execute right now
3396         \fi
3397         \AtBeginDocument{%
3398             \bbl@patchfont{\bbl@xenohyph}%
3399             {\expandafter\select@language\expandafter{\language}}}%
3400     \fi}}%
3401 \fi
3402 \bbl@csarg\bbl@toglobal{lsys@#1}}

```

4.23. Numerals

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

```

3403 \def\bbl@setdigits#1#2#3#4#5{%
3404     \bbl@exp{%
3405         \def<\language\name digits>###1%           i.e., \langdigits
3406         \<bbl@digits@language>###1\\\@nil}%
3407         \let<bbl@cntr@digits@language>\<\language\name digits>%
3408         \def<\language\name counter>###1%           i.e., \langcounter

```

[illegible]

```

3434 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3435   \ifx\\#1%           % \\ before, in case #1 is multiletter
3436     \bbl@exp{%
3437       \def\\bbl@tempa####1{%
3438         \<ifcase>####1space\the\toks@\\<else>\\@ctrerr\<fi>}}%
3439   \else
3440     \toks@\expandafter{\the\toks@\\or #1}%
3441     \expandafter\bbl@buildifcase
3442 \fi}

```

```

3443 \newcommand\localenumeral[2]{%
3444   \bbl@ifunset{\bbl@cntr@#1@\language}%
3445     {#2}%
3446     {\bbl@cs{cntr@#1@\language}{#2}}}
3447 \def\bbl@localecntr#1#2\localenumeral{#2}{#1}%
3448 \newcommand\localecounter[2]{%
3449   \expandafter\bbl@localecntr
3450   \expandafter{\number\csname c@#2\endcsname}{#1}}
3451 \def\bbl@alphnumeral#1#2{%
3452   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3453 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3454   \ifcase\car#8\@nil\or    % Currently <10000, but prepared for bigger
3455     \bbl@alphnumeral@ii{#9}000000#1\or
3456     \bbl@alphnumeral@ii{#9}00000#1#2\or
3457     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3458     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3459     \bbl@error{counter-too-large}{>9999}{}}%
3460   \fi}
3461 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3462   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3463     {\bbl@cs{cntr@#1.4@\language}{#5}%
3464     \bbl@cs{cntr@#1.3@\language}{#6}%

```

```

3465 \bbl@cs{cntr@#1.2@\languagename}#7%
3466 \bbl@cs{cntr@#1.1@\languagename}#8%
3467 \ifnum#6#7#8>\z@
3468 \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3469 {\bbl@cs{cntr@#1.S.321@\languagename}}%
3470 \fi}%
3471 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}

```

Some Chinese counters follow special rules.

```

3472 \def\bbl@zhnumeral#1#2{%
3473 \ifnum#1<10 \bbl@zhnum000#1{#2}%
3474 \else\ifnum#1<100 \bbl@zhnum00#1{#2}%
3475 \else\ifnum#1<1000 \bbl@zhnum0#1{#2}%
3476 \else\ifnum#1<10000 \bbl@zhnum#1{#2}%
3477 \else\bbl@error{counter-too-large}{#1}{}{}%
3478 \fi\fi\fi\fi}
3479 \def\bbl@zhnum#1#2#3#4#5{%
3480 \ifnum\numexpr#1#2#3#4\relax=\z@
3481 \localenumeral{#5.C}\@ne
3482 \else
3483 \ifnum#1>\z@\localenumeral{#5.D}{#1}\localenumeral{#5.C}5\fi % Thousands
3484 \ifnum#2>\z@\localenumeral{#5.D}{#2}\localenumeral{#5.C}4\fi % Hundreds
3485 \else\ifnum#1>\z@\ifnum#3#4>\z@\localenumeral{#5.C}\@ne\fi
3486 \fi\fi
3487 \ifnum#3>\z@ % Tens
3488 \ifnum#1#2=\z@\ifnum#3=\@ne\else\localenumeral{#5.D}{#3}\fi % No  for 10-19:
3489 \else\localenumeral{#5.D}{#3}%
3490 \fi
3491 \localenumeral{#5.C}\thr@@
3492 \else
3493 \ifnum#2>\z@\ifnum#4>\z@\localenumeral{#5.C}\@ne\fi\fi
3494 \fi
3495 \ifnum#4>\z@\localenumeral{#5.D}{#4}\fi % Units
3496 \fi}

```

4.24. Casing

```

3497 \newcommand\BabelUppercaseMapping[3]{%
3498 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3499 \newcommand\BabelTitlecaseMapping[3]{%
3500 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3501 \newcommand\BabelLowercaseMapping[3]{%
3502 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

```

The parser for casing and casing. (*variant*).

```

3503 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3504 \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3505 \else
3506 \def\bbl@uftocode#1{\expandafter`\string#1}
3507 \fi
3508 \def\bbl@casemapping#1#2#3{% 1:variant
3509 \def\bbl@tempa##1 ##2{% Loop
3510 \bbl@casemapping@i{##1}%
3511 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3512 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3513 \def\bbl@tempe{0}% Mode (upper/lower...)
3514 \def\bbl@tempc{#3}% Casing list
3515 \expandafter\bbl@tempa\bbl@tempc\@empty}
3516 \def\bbl@casemapping@i#1{%
3517 \def\bbl@tempb{#1}%
3518 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3519 \@nameuse{regex_replace_all:nnN}%
3520 {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\0}}\bbl@tempb
3521 \else

```

```

3522 \nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb
3523 \fi
3524 \expandafter\bbl@casemapping@ii\bbl@tempb\@@
3525 \def\bbl@casemapping@ii#1#2#3\@@{%
3526 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3527 \ifin@
3528 \edef\bbl@tempe{%
3529 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3530 \else
3531 \ifcase\bbl@tempe\relax
3532 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3533 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3534 \or
3535 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3536 \or
3537 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3538 \or
3539 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3540 \fi
3541 \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.

```

3542 \def\bbl@localeinfo#1#2{%
3543 \bbl@ifunset{\bbl@info@#2}{#1}%
3544 {\bbl@ifunset{\bbl@csname \bbl@info@#2\endcsname @\languagename}{#1}%
3545 {\bbl@cs{\csname \bbl@info@#2\endcsname @\languagename}}}}
3546 \newcommand\localeinfo[1]{%
3547 \ifx*#1\@empty
3548 \bbl@afterelse\bbl@localeinfo{%
3549 \else
3550 \bbl@localeinfo
3551 {\bbl@error{no-ini-info}{}}{}%
3552 {#1}%
3553 \fi}
3554 % \@namedef{\bbl@info@name.locale}{lcname}
3555 \@namedef{\bbl@info@tag.ini}{lini}
3556 \@namedef{\bbl@info@name.english}{elname}
3557 \@namedef{\bbl@info@name.opentype}{lname}
3558 \@namedef{\bbl@info@tag.bcp47}{tbcp}
3559 \@namedef{\bbl@info@language.tag.bcp47}{lbcp}
3560 \@namedef{\bbl@info@tag.opentype}{lotf}
3561 \@namedef{\bbl@info@script.name}{esname}
3562 \@namedef{\bbl@info@script.name.opentype}{sname}
3563 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
3564 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3565 \@namedef{\bbl@info@region.tag.bcp47}{rbcp}
3566 \@namedef{\bbl@info@variant.tag.bcp47}{vbcp}
3567 \@namedef{\bbl@info@extension.t.tag.bcp47}{extt}
3568 \@namedef{\bbl@info@extension.u.tag.bcp47}{extu}
3569 \@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.

```

3570 <<*More package options>> ≡
3571 \DeclareOption{ensureinfo=off}{}
3572 <</More package options>>
3573 \let\BabelEnsureInfo\relax

```

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

```

3574 \newcommand\getlocaleproperty{%
3575 \ifstar\bbl@getproperty@{\bbl@getproperty@x}

```

```

3576 \def\bbl@getproperty@s#1#2#3{%
3577   \let#1\relax
3578   \def\bbl@elt##1##2##3{%
3579     \bbl@ifsamestring{##1/##2}{#3}%
3580     {\providecommand#1{##3}%
3581      \def\bbl@elt####1####2####3{}}}%
3582   {}}%
3583   \bbl@cs{inidata@#2}}%
3584 \def\bbl@getproperty@x#1#2#3{%
3585   \bbl@getproperty@s{#1}{#2}{#3}%
3586   \ifx#1\relax
3587     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3588   \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`.

```

3589 \let\bbl@ini@loaded\@empty
3590 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3591 \def\ShowLocaleProperties#1{%
3592   \typeout{}%
3593   \typeout{*** Properties for language '#1' ***}%
3594   \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3595   \@nameuse{\bbl@inidata@#1}%
3596   \typeout{*****}}

```

4.26. BCP 47 related commands

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

```

3597 \newif\ifbbl@bcppallowed
3598 \bbl@bcppallowedfalse
3599 \def\bbl@autoload@options{@import}
3600 \def\bbl@provide@locale{%
3601   \ifx\babelprovide\undefined
3602     \bbl@error{base-on-the-fly}{#1}{#2}{#3}%
3603   \fi
3604   \let\bbl@auxname\language
3605   \ifbbl@bcptoname
3606     \bbl@ifunset{\bbl@bcp@map@\language}{#1}{#2}{#3} Move uplevel??
3607     {\edef\language{\@nameuse{\bbl@bcp@map@\language}}}%
3608     \let\locale\language
3609   \fi
3610   \ifbbl@bcppallowed
3611     \expandafter\ifx\csname date\language\endcsname\relax
3612       \expandafter
3613       \bbl@bcplookup{\language}%-\@empty-\@empty-\@empty\@@
3614       \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3615         \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3616         \let\locale\language
3617         \expandafter\ifx\csname date\language\endcsname\relax
3618           \let\bbl@initoload\bbl@bcp
3619           \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3620           \let\bbl@initoload\relax
3621         \fi
3622         \bbl@csarg\xdef{\bbl@bcp@\bbl@bcp}{\locale}%
3623       \fi
3624     \fi
3625   \fi

```

```

3626 \expandafter\ifx\csname date\language\endcsname\relax
3627 \IfFileExists{babel-\language.tex}%
3628 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3629 {}%
3630 \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.

```

3631 \providecommand\BCPdata{}
3632 \ifx\renewcommand\undefined\else
3633 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}%
3634 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3635 \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3636 {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3637 {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3638 \def\bbl@bcpdata@ii#1#2{%
3639 \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3640 {\bbl@error{unknown-ini-field}{#1}{}}}%
3641 {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3642 {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3643 \fi
3644 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3645 \@namedef{bbl@info@tag.tag.bcp47}{tbc} % For \BCPdata

```

5. Adjusting the Babel behavior

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

```

3646 \newcommand\babeladjust[1]{%
3647 \bbl@forkv{#1}{%
3648 \bbl@ifunset{bbl@ADJ@##1@##2}%
3649 {\bbl@cs{ADJ@##1}{##2}}%
3650 {\bbl@cs{ADJ@##1@##2}}}%
3651 %
3652 \def\bbl@adjust@lua#1#2{%
3653 \ifvmode
3654 \ifnum\currentgrouplevel=\z@
3655 \directlua{ Babel.#2 }%
3656 \expandafter\expandafter\expandafter@gobble
3657 \fi
3658 \fi
3659 {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3660 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3661 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3662 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3663 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3664 \@namedef{bbl@ADJ@bidi.text@on}{%
3665 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3666 \@namedef{bbl@ADJ@bidi.text@off}{%
3667 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3668 \@namedef{bbl@ADJ@bidi.math@on}{%
3669 \let\bbl@noamsmath\@empty}
3670 \@namedef{bbl@ADJ@bidi.math@off}{%
3671 \let\bbl@noamsmath\relax}
3672 %
3673 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3674 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3675 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3676 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3677 %

```

```

3678 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3679   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3680 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3681   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3682 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3683   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3684 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3685   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3686 \@namedef{bbl@ADJ@justify.arabic@on}{%
3687   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3688 \@namedef{bbl@ADJ@justify.arabic@off}{%
3689   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3690 %
3691 \def\bbl@adjust@layout#1{%
3692   \ifvmode
3693     #1%
3694     \expandafter\@gobble
3695   \fi
3696   {\bbl@error{layout-only-vertical}{}}{}}}% Gobbled if everything went ok.
3697 \@namedef{bbl@ADJ@layout.tabular@on}{%
3698   \ifnum\bbl@tabular@mode=\tw@
3699     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3700   \else
3701     \chardef\bbl@tabular@mode\@ne
3702   \fi}
3703 \@namedef{bbl@ADJ@layout.tabular@off}{%
3704   \ifnum\bbl@tabular@mode=\tw@
3705     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3706   \else
3707     \chardef\bbl@tabular@mode\@z@
3708   \fi}
3709 \@namedef{bbl@ADJ@layout.lists@on}{%
3710   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3711 \@namedef{bbl@ADJ@layout.lists@off}{%
3712   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3713 %
3714 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3715   \bbl@bcppallowedtrue}
3716 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3717   \bbl@bcppallowedfalse}
3718 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3719   \def\bbl@bcp@prefix{#1}}
3720 \def\bbl@bcp@prefix{bcp47-}
3721 \@namedef{bbl@ADJ@autoload.options#1}{%
3722   \def\bbl@autoload@options{#1}}
3723 \def\bbl@autoload@bcptoptions{import}
3724 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3725   \def\bbl@autoload@bcptoptions{#1}}
3726 \newif\ifbbl@bcptoname
3727 %
3728 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3729   \bbl@bcptonametrue}
3730 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3731   \bbl@bcptonamefalse}
3732 %
3733 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3734   \directlua{ Babel.ignore_pre_char = function(node)
3735     return (node.lang == \the\csname \nohyphenation\endcsname)
3736   end }}
3737 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3738   \directlua{ Babel.ignore_pre_char = function(node)
3739     return false
3740   end }}

```

```

3741 %
3742 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3743   \def\bbl@ignoreinterchar{%
3744     \ifnum\language=\l@nohyphenation
3745       \expandafter\@gobble
3746     \else
3747       \expandafter\@firstofone
3748     \fi}}
3749 \@namedef{bbl@ADJ@interchar.disable@off}{%
3750   \let\bbl@ignoreinterchar\@firstofone}
3751 %
3752 \@namedef{bbl@ADJ@select.write@shift}{%
3753   \let\bbl@restorelastskip\relax
3754   \def\bbl@savelastskip{%
3755     \let\bbl@restorelastskip\relax
3756     \ifvmode
3757       \ifdim\lastskip=\z@
3758         \let\bbl@restorelastskip\nobreak
3759       \else
3760         \bbl@exp{%
3761           \def\\bbl@restorelastskip{%
3762             \skip@=\the\lastskip
3763             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3764         \fi
3765       \fi}}
3766 \@namedef{bbl@ADJ@select.write@keep}{%
3767   \let\bbl@restorelastskip\relax
3768   \let\bbl@savelastskip\relax}
3769 \@namedef{bbl@ADJ@select.write@omit}{%
3770   \AddBabelHook{babel-select}{beforestart}{%
3771     \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3772   \let\bbl@restorelastskip\relax
3773   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3774 \@namedef{bbl@ADJ@select.encoding@off}{%
3775   \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.

```

3776 << *More package options >> ≡
3777 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3778 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3779 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3780 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3781 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3782 << /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.

```

3783 \bbl@trace{Cross referencing macros}
3784 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3785   \def\@newl@bel#1#2#3{%

```

```

3786 {\@safe@activestrue
3787 \bbl@ifunset{#1@#2}%
3788 \relax
3789 {\gdef\@multiplelabels{%
3790 \@latex@warning@no@line{There were multiply-defined labels}}%
3791 \@latex@warning@no@line{Label `#2' multiply defined}}%
3792 \global\@namedef{#1@#2}{#3}}

```

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

```

3793 \CheckCommand*\@testdef[3]{%
3794 \def\reserved@a{#3}%
3795 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3796 \else
3797 \@tempswatrue
3798 \fi}

```

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

```

3799 \def\@testdef#1#2#3{%
3800 \@safe@activestrue
3801 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3802 \def\bbl@tempb{#3}%
3803 \@safe@activestrue
3804 \ifx\bbl@tempa\relax
3805 \else
3806 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3807 \fi
3808 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3809 \ifx\bbl@tempa\bbl@tempb
3810 \else
3811 \@tempswatrue
3812 \fi}
3813 \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.

```

3814 \bbl@xin@{R}\bbl@opt@safe
3815 \ifin@
3816 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3817 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3818 {\expandafter\strip@prefix\meaning\ref}%
3819 \ifin@
3820 \bbl@redefine\@kernel@ref#1{%
3821 \@safe@activestrue\org@@kernel@ref{#1}\@safe@activestrue}
3822 \bbl@redefine\@kernel@pageref#1{%
3823 \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activestrue}
3824 \bbl@redefine\@kernel@sref#1{%
3825 \@safe@activestrue\org@@kernel@sref{#1}\@safe@activestrue}
3826 \bbl@redefine\@kernel@spageref#1{%
3827 \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activestrue}
3828 \else
3829 \bbl@redefineroobust\ref#1{%
3830 \@safe@activestrue\org@ref{#1}\@safe@activestrue}
3831 \bbl@redefineroobust\pageref#1{%
3832 \@safe@activestrue\org@pageref{#1}\@safe@activestrue}
3833 \fi

```

```

3834 \else
3835   \let\org@ref\ref
3836   \let\org@pageref\pageref
3837 \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.

```

3838 \bbl@xin@{B}\bbl@opt@safe
3839 \ifin@
3840   \bbl@redefine\@citex[#1]#2{%
3841     \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activfalse
3842     \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.)

```

3843 \AtBeginDocument{%
3844   \@ifpackageloaded{natbib}{%
3845     \def\@citex[#1]#2#3{%
3846       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activfalse
3847       \org@@citex[#1]#2{\bbl@tempa}}%
3848   }{}}

```

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

```

3849 \AtBeginDocument{%
3850   \@ifpackageloaded{cite}{%
3851     \def\@citex[#1]#2{%
3852       \@safe@activetrue\org@@citex[#1]#2\@safe@activfalse}%
3853   }{}}

```

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

```

3854 \bbl@redefine\nocite#1{%
3855   \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

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

```

3856 \bbl@redefine\bibcite{%
3857   \bbl@cite@choice
3858   \bibcite}

```

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

```

3859 \def\bbl@bibcite#1#2{%
3860   \org@bibcite{#1}\@safe@activfalse#2}}

```

\bbl@cite@choice The macro \bbl@cite@choice determines which definition of \bibcite is needed. First we give \bibcite its default definition.

```

3861 \def\bbl@cite@choice{%
3862   \global\let\bibcite\bbl\bibcite
3863   \@ifpackageloaded{natbib}{\global\let\bibcite\org\bibcite}{}%
3864   \@ifpackageloaded{cite}{\global\let\bibcite\org\bibcite}{}%
3865   \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.

```

3866 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3867 \bbl@redefine\@bibitem#1{%
3868   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
3869 \else
3870   \let\org@nocite\nocite
3871   \let\org@citex\@citex
3872   \let\org@bibcite\bibcite
3873   \let\org@bibitem\@bibitem
3874 \fi

```

5.2. Layout

```

3875 \newcommand\BabelPatchSection[1]{%
3876   \ifundefined{#1}{}{%
3877     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3878     \@namedef{#1}{%
3879       \ifstar{\bbl@presec@#1}%
3880       {\@dblarg{\bbl@presec@x{#1}}}}%
3881 \def\bbl@presec@x#1[#2]#3{%
3882   \bbl@exp{%
3883     \\select@language@x{\bbl@main@language}%
3884     \\bbl@cs{sspre@#1}%
3885     \\bbl@cs{ss@#1}%
3886     [\\foreignlanguage{\language@name}{\unexpanded{#2}}}%
3887     {\\foreignlanguage{\language@name}{\unexpanded{#3}}}%
3888     \\select@language@x{\language@name}}%
3889 \def\bbl@presec@s#1#2{%
3890   \bbl@exp{%
3891     \\select@language@x{\bbl@main@language}%
3892     \\bbl@cs{sspre@#1}%
3893     \\bbl@cs{ss@#1}*%
3894     {\\foreignlanguage{\language@name}{\unexpanded{#2}}}%
3895     \\select@language@x{\language@name}}%
3896 %
3897 \IfBabelLayout{sectioning}%
3898   {\BabelPatchSection{part}%
3899    \BabelPatchSection{chapter}%
3900    \BabelPatchSection{section}%
3901    \BabelPatchSection{subsection}%
3902    \BabelPatchSection{subsubsection}%
3903    \BabelPatchSection{paragraph}%
3904    \BabelPatchSection{subparagraph}%
3905    \def\babel@toc#1{%
3906      \select@language@x{\bbl@main@language}}}%
3907 \IfBabelLayout{captions}%
3908   {\BabelPatchSection{caption}}}%

```

\BabelFootnote Footnotes.

```

3909 \bbl@trace{Footnotes}

```

```

3910 \def\bbl@footnote#1#2#3{%
3911   \@ifnextchar[%
3912     {\bbl@footnote@o{#1}{#2}{#3}}%
3913     {\bbl@footnote@x{#1}{#2}{#3}}}
3914 \long\def\bbl@footnote@x#1#2#3#4{%
3915   \bgroup
3916     \select@language@x{\bbl@main@language}%
3917     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3918   \egroup}
3919 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3920   \bgroup
3921     \select@language@x{\bbl@main@language}%
3922     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3923   \egroup}
3924 \def\bbl@footnotetext#1#2#3{%
3925   \@ifnextchar[%
3926     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3927     {\bbl@footnotetext@x{#1}{#2}{#3}}}
3928 \long\def\bbl@footnotetext@x#1#2#3#4{%
3929   \bgroup
3930     \select@language@x{\bbl@main@language}%
3931     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3932   \egroup}
3933 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3934   \bgroup
3935     \select@language@x{\bbl@main@language}%
3936     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3937   \egroup}
3938 \def\BabelFootnote#1#2#3#4{%
3939   \ifx\bbl@fn@footnote\undefined
3940     \let\bbl@fn@footnote\footnote
3941   \fi
3942   \ifx\bbl@fn@footnotetext\undefined
3943     \let\bbl@fn@footnotetext\footnotetext
3944   \fi
3945   \bbl@ifblank{#2}%
3946     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3947     \@namedef{\bbl@stripslash#1text}%
3948     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3949     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3950     \@namedef{\bbl@stripslash#1text}%
3951     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3952 \IfBabelLayout{footnotes}%
3953 {\let\bbl@OL@footnote\footnote
3954   \BabelFootnote\footnote\languagesname{}}}%
3955 {\BabelFootnote\localfootnote\languagesname{}}}%
3956 {\BabelFootnote\mainfootnote{}}}%
3957 {}

```

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.

```

3958 \bbl@trace{Marks}
3959 \IfBabelLayout{sectioning}
3960 {\ifx\bbl@opt@headfoot\@nnil
3961   \g@addto@macro\resetactivechars{%
3962     \set@typeset@protect
3963     \expandafter\select@language@x\expandafter{\bbl@main@language}%

```

```

3964 \let\protect\noexpand
3965 \ifcase\bbbl@bidimode\else % Only with bidi. See also above
3966 \edef\thepage{%
3967 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3968 \fi}%
3969 \fi}
3970 {\ifbbl@single\else
3971 \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3972 \markright#1{%
3973 \bbl@ifblank{#1}%
3974 {\org@markright{}}}%
3975 {\toks@{#1}%
3976 \bbl@exp{%
3977 \org@markright{\protect\foreignlanguage{\language}\language}%
3978 \protect\bbl@restore@actives\the\toks@}}}%

```

\markboth

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

```

3979 \ifx\@mkboth\markboth
3980 \def\bbl@tempc{\let\@mkboth\markboth}%
3981 \else
3982 \def\bbl@tempc{}%
3983 \fi
3984 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3985 \markboth#1#2{%
3986 \protected@edef\bbl@tempb##1{%
3987 \protect\foreignlanguage
3988 {\language}\protect\bbl@restore@actives##1}%
3989 \bbl@ifblank{#1}%
3990 {\toks@{}}%
3991 {\toks@\expandafter{\bbl@tempb{#1}}}%
3992 \bbl@ifblank{#2}%
3993 {\@temptokena{}}%
3994 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3995 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3996 \bbl@tempc
3997 \fi} % end ifbbl@single, end \IfBabelLayout

```

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.

```

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

```

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`.

```

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

```

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

```

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

```

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.

```

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

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

```

4047 \def\substitutefontfamily#1#2#3{%
4048   \lowercase{\immediate\openout15=#1#2.fd\relax}%
4049   \immediate\write15{%
4050     \string\ProvidesFile{#1#2.fd}%
4051     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
4052     \space generated font description file]^J
4053     \string\DeclareFontFamily{#1}{#2}{}}^J
4054     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
4055     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
4056     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
4057     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
4058     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
4059     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
4060     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
4061     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
4062   }%
4063   \closeout15
4064 }
4065 \@onlypreamble\substitutefontfamily
```

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

```

4066 \bbl@trace{Encoding and fonts}
4067 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
4068 \newcommand\BabelNonText{TS1,T3,TS3}
4069 \let\org@TeX\TeX
4070 \let\org@LaTeX\LaTeX
4071 \let\ensureascii@firstofone
4072 \let\asciiencoding\empty
4073 \AtBeginDocument{%
4074   \def\@elt#1{,#1,}%
4075   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4076   \let\@elt\relax
4077   \let\bbl@tempb\empty
4078   \def\bbl@tempc{OT1}%
4079   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
4080     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
4081   \bbl@foreach\bbl@tempa{%
```

```

4082 \bbl@xin@{,#1,}{,\BabelNonASCII,}%
4083 \ifin@
4084 \def\bbl@tempb{#1}% Store last non-ascii
4085 \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
4086 \ifin@else
4087 \def\bbl@tempc{#1}% Store last ascii
4088 \fi
4089 \fi}%
4090 \ifx\bbl@tempb\@empty\else
4091 \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
4092 \ifin@else
4093 \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
4094 \fi
4095 \let\asciencoding\bbl@tempc
4096 \renewcommand\ensureasci[1]{%
4097 {\fontencoding{\asciencoding}\selectfont#1}}%
4098 \DeclareTextCommandDefault{\TeX}{\ensureasci{\org@TeX}}%
4099 \DeclareTextCommandDefault{\LaTeX}{\ensureasci{\org@LaTeX}}%
4100 \fi}

```

Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at `\begin{document}`, which latin fontencoding to use.

Latinencoding When text is being typeset in an encoding other than ‘latin’ (OT1 or T1), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

```

4101 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}

```

But this might be overruled with a later loading of the package fontenc. Therefore we check at the execution of `\begin{document}` whether it was loaded with the T1 option. The normal way to do this (using `\@ifpackageloaded`) is disabled for this package. Now we have to revert to parsing the internal macro `\@filelist` which contains all the filenames loaded.

```

4102 \AtBeginDocument{%
4103 \@ifpackageloaded{fontspec}%
4104 {\xdef\latinencoding{%
4105 \ifx\UTFencname\@undefined
4106 EU\ifcase\bbl@engine\or2\or1\fi
4107 \else
4108 \UTFencname
4109 \fi}}%
4110 {\gdef\latinencoding{OT1}%
4111 \ifx\cf@encoding\bbl@t@one
4112 \xdef\latinencoding{\bbl@t@one}%
4113 \else
4114 \def\@elt#1{,#1,}%
4115 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4116 \let\@elt\relax
4117 \bbl@xin@{,T1,}\bbl@tempa
4118 \ifin@
4119 \xdef\latinencoding{\bbl@t@one}%
4120 \fi
4121 \fi}}

```

Latintext Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

```

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

```

Textlatin This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

```

4125 \ifx\@undefined\DeclareTextFontCommand

```

```

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

```

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

```

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

```

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

```

4131 \bbl@trace{Loading basic (internal) bidi support}
4132 \ifodd\bbl@engine
4133 \else % Any xe+lua bidi
4134 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
4135 \bbl@error{bidi-only-lua}{-}{-}%
4136 \let\bbl@beforeforeign\leavevmode
4137 \AtEndOfPackage{%
4138 \EnableBabelHook{babel-bidi}%
4139 \bbl@xebidipar}
4140 \fi\fi
4141 \def\bbl@loadxebidi#1{%
4142 \ifx\RTLfootnotetext\undefined
4143 \AtEndOfPackage{%
4144 \EnableBabelHook{babel-bidi}%
4145 \ifx\fontspec\undefined
4146 \usepackage{fontspec}% bidi needs fontspec
4147 \fi
4148 \usepackage#1{bidi}%
4149 \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4150 \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4151 \ifnum\@nameuse{\bbl@wdir\@languagename}=\tw@ % 'AL' bidi
4152 \bbl@digitsdotdash % So ignore in 'R' bidi
4153 \fi}}%
4154 \fi}
4155 \ifnum\bbl@bidimode>200 % Any xe bidi=
4156 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4157 \bbl@tentative{bidi=bidi}
4158 \bbl@loadxebidi{}
4159 \or
4160 \bbl@loadxebidi{[rldocument]}
4161 \or
4162 \bbl@loadxebidi{}

```

```

4163 \fi
4164 \fi
4165 \fi
4166 \ifnum\bbl@bidimode=\@ne % bidi=default
4167 \let\bbl@beforeforeign\leavevmode
4168 \ifodd\bbl@engine % lua
4169 \newattribute\bbl@attr@dir
4170 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4171 \bbl@exp{\output{\bodydir\pagedir\the\output}}
4172 \fi
4173 \AtEndOfPackage{%
4174 \EnableBabelHook{babel-bidi}% pdf/lua/x
4175 \ifodd\bbl@engine\else % pdf/x
4176 \bbl@xebidipar
4177 \fi}
4178 \fi

```

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

```

4179 \bbl@trace{Macros to switch the text direction}
4180 \def\bbl@alscripts{%
4181 ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
4182 \def\bbl@rscripts{%
4183 Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4184 Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
4185 Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
4186 Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
4187 Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
4188 Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
4189 Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
4190 Meroitic,N'Ko,Orkhon,Todhri}
4191 %
4192 \def\bbl@provide@dirs#1{%
4193 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4194 \ifin@
4195 \global\bbl@csarg\chardef{wdir@#1}\@ne
4196 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4197 \ifin@
4198 \global\bbl@csarg\chardef{wdir@#1}\tw@
4199 \fi
4200 \else
4201 \global\bbl@csarg\chardef{wdir@#1}\z@
4202 \fi
4203 \ifodd\bbl@engine
4204 \bbl@csarg\ifcase{wdir@#1}%
4205 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4206 \or
4207 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4208 \or
4209 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4210 \fi
4211 \fi}
4212 %
4213 \def\bbl@switchdir{%
4214 \bbl@ifunset\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
4215 \bbl@ifunset\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%
4216 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4217 \def\bbl@setdirs#1{%
4218 \ifcase\bbl@select@type
4219 \bbl@bodydir{#1}%
4220 \bbl@pardir{#1}% <- Must precede \bbl@texdir
4221 \fi

```

```

4222 \bbl@textdir{#1}}
4223 \ifnum\bbl@bidimode>\z@
4224 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4225 \DisableBabelHook{babel-bidi}
4226 \fi

Now the engine-dependent macros.

4227 \ifodd\bbl@engine % luatex=1
4228 \else % pdftex=0, xetex=2
4229 \newcount\bbl@dirlevel
4230 \chardef\bbl@thetextdir\z@
4231 \chardef\bbl@thepardir\z@
4232 \def\bbl@textdir#1{%
4233 \ifcase#1\relax
4234 \chardef\bbl@thetextdir\z@
4235 \@nameuse{setlatin}%
4236 \bbl@textdir@i\beginL\endL
4237 \else
4238 \chardef\bbl@thetextdir\@ne
4239 \@nameuse{setnonlatin}%
4240 \bbl@textdir@i\beginR\endR
4241 \fi}
4242 \def\bbl@textdir@i#1#2{%
4243 \ifhmode
4244 \ifnum\currentgrouplevel>\z@
4245 \ifnum\currentgrouplevel=\bbl@dirlevel
4246 \bbl@error{multiple-bidi}{\}\}\}%
4247 \bgroup\aftergroup#2\aftergroup\egroup
4248 \else
4249 \ifcase\currentgrouptype\or % 0 bottom
4250 \aftergroup#2% 1 simple {}
4251 \or
4252 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4253 \or
4254 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4255 \or\or\or % vbox vtop align
4256 \or
4257 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4258 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4259 \or
4260 \aftergroup#2% 14 \begingroup
4261 \else
4262 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4263 \fi
4264 \fi
4265 \bbl@dirlevel\currentgrouplevel
4266 \fi
4267 #1%
4268 \fi}
4269 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4270 \let\bbl@bodydir\@gobble
4271 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

4272 \def\bbl@xebidipar{%
4273 \let\bbl@xebidipar\relax
4274 \TeXeTstate\@ne
4275 \def\bbl@xeeverypar{%
4276 \ifcase\bbl@thepardir
4277 \ifcase\bbl@thetextdir\else\beginR\fi
4278 \else
4279 {\setbox\z@\lastbox\beginR\box\z@}%

```

```

4280     \fi}%
4281     \AddToHook{para/begin}{\bbl@xeeverypar}}
4282     \ifnum\bbl@bidimode>200 % Any xe bidi=
4283     \let\bbl@textdir@i@gobbletwo
4284     \let\bbl@xebidipar@empty
4285     \AddBabelHook{bidi}{foreign}{%
4286       \ifcase\bbl@thetextdir
4287       \BabelWrapText{\LR{##1}}%
4288       \else
4289       \BabelWrapText{\RL{##1}}%
4290     \fi}
4291     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4292     \fi
4293 \fi

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

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

```

4301 \bbl@trace{Local Language Configuration}
4302 \ifx\loadlocalcfg\undefined
4303   \@ifpackagewith{babel}{noconfigs}%
4304   {\let\loadlocalcfg@gobble}%
4305   {\def\loadlocalcfg#1{%
4306     \InputIfFileExists{#1.cfg}%
4307     {\typeout{*****^J%
4308               * Local config file #1.cfg used^^J%
4309               *}}%
4310     \@empty}}
4311 \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).

```

4312 \bbl@trace{Language options}
4313 \def\BabelDefinitionFile#1#2#3{}
4314 \let\bbl@afterlang\relax
4315 \let\BabelModifiers\relax
4316 \let\bbl@loaded@empty
4317 \def\bbl@load@language#1{%
4318   \InputIfFileExists{#1.ldf}%
4319   {\edef\bbl@loaded{\CurrentOption
4320     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
4321     \expandafter\let\expandafter\bbl@afterlang
4322     \csname\CurrentOption.ldf-h@k\endcsname
4323     \expandafter\let\expandafter\BabelModifiers
4324     \csname\bbl@mod@\CurrentOption\endcsname

```

```

4325 \bbl@exp{\AtBeginDocument{%
4326 \bbl@usehooks@lang{\CurrentOption}{\beginDocument}{\CurrentOption}}}%
4327 {\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 `DocumentMetadata` we also force it with `\foreignlanguage` (this is also done in `bidi` texts).

```

4328 \ifx\GetDocumentProperties\undefined\else
4329 \let\bbl@beforeforeign\leavevmode
4330 \edef\bbl@tempa{\GetDocumentProperties{document/other-languages}}%
4331 \let\bbl@collect@other\@empty
4332 \bbl@foreach\bbl@tempa{%
4333 \edef\bbl@metalang{#1}%
4334 \ifx\bbl@metalang\@empty\else
4335 \begingroup
4336 \expandafter
4337 \bbl@bcplookup{\bbl@metalang}%-\@empty-\@empty-\@empty@@
4338 \ifx\bbl@bcp\relax
4339 \ifx\bbl@opt@main\@nnil
4340 \bbl@error{no-locale-for-meta}{\bbl@metalang}{}{}%
4341 \fi
4342 \else
4343 \bbl@read@ini{\bbl@bcp}\m@ne
4344 \xdef\bbl@collect@other{\bbl@collect@other,\language}%
4345 \GenericInfo{}{(babel) \spaces\@spaces\@spaces
4346 Passing '\language' to babel\@gobble}%
4347 \fi
4348 \endgroup
4349 \fi}
4350 \ifx\bbl@collect@other\@empty\else
4351 \xdef\bbl@language@opts{\bbl@collect@other,\bbl@language@opts}%
4352 \fi
4353 \edef\bbl@metalang{\GetDocumentProperties{document/language}}%
4354 \ifx\bbl@metalang\@empty\else
4355 \begingroup
4356 \expandafter
4357 \bbl@bcplookup{\bbl@metalang}%-\@empty-\@empty-\@empty@@
4358 \ifx\bbl@bcp\relax
4359 \ifx\bbl@opt@main\@nnil
4360 \bbl@error{no-locale-for-meta}{\bbl@metalang}{}{}%
4361 \fi
4362 \else
4363 \bbl@read@ini{\bbl@bcp}\m@ne
4364 \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4365 \ifx\bbl@opt@main\@nnil
4366 \global\let\bbl@opt@main\language
4367 \fi
4368 \GenericInfo{}{(babel) \spaces\@spaces\@spaces
4369 Passing '\language' to babel\@gobble}%
4370 \fi
4371 \endgroup
4372 \fi
4373 \fi
4374 \ifx\bbl@opt@config\@nnil
4375 \ifpackagewith{babel}{noconfigs}{}%
4376 {\InputIfFileExists{bblopts.cfg}%

```

```

4377      {\bbl@info{Configuration files are deprecated, as\\%
4378              they can break document portability.\\%
4379              Reported}%
4380      \typeout{*****^J%
4381              * Local config file bblopts.cfg used^J%
4382              *}%
4383      {}}%
4384 \else
4385   \InputIfFileExists{\bbl@opt@config.cfg}%
4386   {\bbl@info{Configuration files are deprecated, as\\%
4387           they can break document portability.\\%
4388           Reported}%
4389   \typeout{*****^J%
4390           * Local config file \bbl@opt@config.cfg used^J%
4391           *}%
4392   {\bbl@error{config-not-found}}{}{}}%
4393 \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 contain 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 loaded directly (with `@import`).

```

4394 \def\BabelBeforeIni#1#2{%
4395   \def\bbl@tempa{\@m}% <- Default if no \BDefFile
4396   \let\bbl@tempb\@empty
4397   #2%
4398   \edef\bbl@toload{%
4399     \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4400     \bbl@toload@last}%
4401   \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4402 \def\BabelDefinitionFile#1#2#3{%
4403   \def\bbl@tempa{#1}\def\bbl@tempb{#2}%
4404   \@namedef{\bbl@preldf@\CurrentOption}{#3}%
4405   \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.

```

4406 \def\bbl@tempf{,}
4407 \@nameuse{clist_map_inline:Nn}\@raw@classoptionslist{%
4408   \in@{=}{#1}%
4409   \ifin@ \else
4410     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4411   \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.

```

4412 \let\bbl@toload\@empty
4413 \let\bbl@toload@last\@empty
4414 \let\bbl@unkopt\@gobble %% <- Ugly
4415 \edef\bbl@tempc{%
4416   \bbl@tempf,@@,\bbl@language@opts

```

```

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

```

4438 \ifx\bbl@opt@main@nnil
4439   \ifx\bbl@toload@last@empty
4440     \def\bbl@toload@last{0/0//nil//und-x-nil-nil}
4441     \bbl@info{%
4442       You haven't specified a language as a class or package\\%
4443       option. I'll load 'nil'. Reported}
4444     \fi
4445   \else
4446     \let\bbl@tempc@empty
4447     \bbl@foreach\bbl@toload{%
4448       \bbl@xin@{/#1//}{\bbl@opt@main//}{#1}%
4449       \ifin@else
4450         \bbl@add@list\bbl@tempc{#1}%
4451       \fi}
4452     \let\bbl@toload\bbl@tempc
4453   \fi
4454   \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'.

```

4455 \def\AfterBabelLanguage#1{%
4456   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4457 \NewHook{babel/presets}
4458 \UseHook{babel/presets}
4459 %
4460 \let\bbl@tempb@empty
4461 \def\bbl@tempc#1/#2//#3//#4/#5\@@{%
4462   \count@z@
4463   \ifnum#2=\@m % if no \BabelDefinitionFile
4464     \ifnum#1=z@ % not main. -- % if provide=, provide*=!
4465       \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0//#3//#4/#3\@@
4466       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4467     \fi
4468   \else % 10 = main -- % if provide=, provide*=!
4469     \ifodd\bbl@ldfflag\bbl@tempc 10/0//#3//#4/#3\@@
4470     \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4471   \fi

```

```

4472 \fi
4473 \else
4474 \ifnum#1=\z@ % not main
4475 \ifnum\bb@iniflag>\@ne\else % if ø, provide
4476 \ifcase#2\count@\@ne\else\ifcase\bb@engine\count@\@ne\fi\fi
4477 \fi
4478 \else % l0 = main
4479 \ifodd\bb@iniflag\else % if provide+, provide*
4480 \ifcase#2\count@\@ne\else\ifcase\bb@engine\count@\@ne\fi\fi
4481 \fi
4482 \fi
4483 \bb@tempd{#1}{#2}{#3}{#4}{#5}%
4484 \fi}

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

4485 \def\bb@tempd#1#2#3#4#5{%
4486 \DeclareOption{#3}{}%
4487 \ifcase\count@
4488 \bb@exp{\bb@add\bb@tempb{%
4489 \global\let\bb@afterload\@empty
4490 \nameuse{bb@preini#3}%
4491 \bb@ldfinit
4492 \def\CurrentOption{#3}%
4493 \babelprovide[import=#4,\ifnum#1=\z@\else\bb@opt@provide,main\fi]{#3}%
4494 \bb@afterldf
4495 \bb@afterload}}%
4496 \else
4497 \bb@add\bb@tempb{%
4498 \global\let\bb@afterload\@empty
4499 \def\CurrentOption{#3}%
4500 \let\localename\CurrentOption
4501 \let\language\localename
4502 \def\BabelIniTag{#4}%
4503 \nameuse{bb@preldf#3}%
4504 \begingroup
4505 \bb@id@assign
4506 \bb@read@ini{\BabelIniTag}0%
4507 \endgroup
4508 \bb@load@language{#5}%
4509 \bb@afterload}%
4510 \fi}
4511 %
4512 \bb@foreach\bb@toload{\bb@tempc#1\@@}
4513 \bb@tempb
4514 \DeclareOption*{}
4515 \ProcessOptions
4516 %
4517 \bb@exp{%
4518 \AtBeginDocument{\bb@usehooks@lang{/}{begindocument}{}}}%
4519 \def\AfterBabelLanguage{\bb@error{late-after-babel}{}}
4520 \AddToHook{begindocument/end}{%
4521 \bb@info{Main language set to '\mainlocalename'\@gobble}}
4522 </package>

```

6. The kernel of Babel

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

Because plain $\TeX$ users might want to use some of the features of the babel system too, care has to be taken that plain $\TeX$ can process the files. For this reason the current format will have to be

checked in a number of places. Some of the code below is common to plain $\TeX$ and $\LaTeX$, some of it is for the $\LaTeX$ case only.

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

A proxy file for `switch.def`

```
4523 \kernel
4524 \let\bbl@onlyswitch\@empty
4525 \input babel.def
4526 \let\bbl@onlyswitch\@undefined
4527 \kernel
```

7. Error messages

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

```
4528 \errors
4529 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4530 \catcode`\:=12 \catcode`\.,=12 \catcode`\.=12 \catcode`\-=12
4531 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4532 \catcode`\@=11 \catcode`\^=7
4533 %
4534 \ifx\MessageBreak\@undefined
4535 \gdef\bbl@error@i#1#2{%
4536   \begingroup
4537     \newlinechar=`^^J
4538     \def\{^^J(babel) }%
4539     \errhelp{#2}\errmessage{\{#1}%
4540   \endgroup}
4541 \else
4542 \gdef\bbl@error@i#1#2{%
4543   \begingroup
4544     \def\{\MessageBreak}%
4545     \PackageError{babel}{#1}{#2}%
4546   \endgroup}
4547 \fi
4548 \def\bbl@errmessage#1#2#3{%
4549   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4550     \bbl@error@i{#2}{#3}}
4551 % Implicit #2#3#4:
4552 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4553 %
4554 \bbl@errmessage{not-yet-available}
4555   {Not yet available}%
4556   {Find an armchair, sit down and wait}
4557 \bbl@errmessage{bad-package-option}%
4558   {Bad option '#1=#2'. Either you have misspelled the\\%
4559   key or there is a previous setting of '#1'. Valid\\%
4560   keys are, among others, 'shorthands', 'main', 'bidi',\\%
4561   'strings', 'config', 'headfoot', 'safe', 'math'.}%
4562   {See the manual for further details.}
4563 \bbl@errmessage{base-on-the-fly}
4564   {For a language to be defined on the fly 'base'\\%
4565   is not enough, and the whole package must be\\%
4566   loaded. Either delete the 'base' option or\\%
4567   request the languages explicitly}%
4568   {See the manual for further details.}
4569 \bbl@errmessage{undefined-language}
4570   {You haven't defined the language '#1' yet.\\%
```

```

4571     Perhaps you misspelled it or your installation\\%
4572     is not complete}%
4573     {Your command will be ignored, type <return> to proceed}
4574 \bbl@errmessage{invalid-ini-name}
4575     {'#1' not valid with the 'ini' mechanism.\\%
4576     I think you want '#2' instead. You may continue,\\%
4577     but you should fix the name. See the babel manual\\%
4578     for the available locales with 'provide'}%
4579     {See the manual for further details.}
4580 \bbl@errmessage{shorthand-is-off}
4581     {I can't declare a shorthand turned off (\string#2)}
4582     {Sorry, but you can't use shorthands which have been\\%
4583     turned off in the package options}
4584 \bbl@errmessage{not-a-shorthand}
4585     {The character '\string #1' should be made a shorthand character;\\%
4586     add the command \string\usesshorthands\string{#1\string} to
4587     the preamble.\\%
4588     I will ignore your instruction}%
4589     {You may proceed, but expect unexpected results}
4590 \bbl@errmessage{not-a-shorthand-b}
4591     {I can't switch '\string#2' on or off--not a shorthand\\%
4592     This character is not a shorthand. Maybe you made\\%
4593     a typing mistake?}%
4594     {I will ignore your instruction.}
4595 \bbl@errmessage{unknown-attribute}
4596     {The attribute #2 is unknown for language #1.}%
4597     {Your command will be ignored, type <return> to proceed}
4598 \bbl@errmessage{missing-group}
4599     {Missing group for string \string#1}%
4600     {You must assign strings to some category, typically\\%
4601     captions or extras, but you set none}
4602 \bbl@errmessage{only-lua-xe}
4603     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4604     {Consider switching to these engines.}
4605 \bbl@errmessage{only-lua}
4606     {This macro is available only in LuaLaTeX}%
4607     {Consider switching to that engine.}
4608 \bbl@errmessage{unknown-provide-key}
4609     {Unknown key '#1' in \string\babelprovide}%
4610     {See the manual for valid keys}%
4611 \bbl@errmessage{unknown-mapfont}
4612     {Option '\bbl@KVP@mapfont' unknown for\\%
4613     mapfont. Use 'direction'}%
4614     {See the manual for details.}
4615 \bbl@errmessage{no-ini-file}
4616     {There is no ini file for the requested language\\%
4617     (#1: \language). Perhaps you misspelled it or your\\%
4618     installation is not complete}%
4619     {Fix the name or reinstall babel.}
4620 \bbl@errmessage{digits-is-reserved}
4621     {The counter name 'digits' is reserved for mapping\\%
4622     decimal digits}%
4623     {Use another name.}
4624 \bbl@errmessage{limit-two-digits}
4625     {Currently two-digit years are restricted to the\\5
4626     range 0-9999}%
4627     {There is little you can do. Sorry.}
4628 \bbl@errmessage{counter-too-large}
4629     {Counter too large (#1)}%
4630     {Currently this is the limit.}
4631 \bbl@errmessage{no-ini-info}
4632     {I've found no info for the current locale.\\%
4633     The corresponding ini file has not been loaded\\%

```

```

4634     Perhaps it doesn't exist}%
4635     {See the manual for details.}
4636 \bbl@errmessage{unknown-ini-field}
4637     {Unknown field '#1' in \string\BCPdata.\\%
4638     Perhaps you misspelled it}%
4639     {See the manual for details.}
4640 \bbl@errmessage{unknown-locale-key}
4641     {Unknown key for locale '#2':\\%
4642     #3\\%
4643     \string#1 will be set to \string\relax}%
4644     {Perhaps you misspelled it.}%
4645 \bbl@errmessage{adjust-only-vertical}
4646     {Currently, #1 related features can be adjusted only\\%
4647     in the main vertical list}%
4648     {Maybe things change in the future, but this is what it is.}
4649 \bbl@errmessage{layout-only-vertical}
4650     {Currently, layout related features can be adjusted only\\%
4651     in vertical mode}%
4652     {Maybe things change in the future, but this is what it is.}
4653 \bbl@errmessage{bidi-only-lua}
4654     {The bidi method 'basic' is available only in\\%
4655     luatex. I'll continue with 'bidi=default', so\\%
4656     expect wrong results.\\%
4657     Suggested actions:\\%
4658     * If possible, switch to luatex, as xetex is not\\%
4659     recommend anymore.\\
4660     * If you can't, try 'bidi=bidi' with xetex.\\%
4661     * With pdftex, only 'bidi=default' is available.}%
4662     {See the manual for further details.}
4663 \bbl@errmessage{multiple-bidi}
4664     {Multiple bidi settings inside a group\\%
4665     I'll insert a new group, but expect wrong results.\\%
4666     Suggested action:\\%
4667     * Add a new group where appropriate.}
4668     {See the manual for further details.}
4669 \bbl@errmessage{unknown-package-option}
4670     {Unknown option '\CurrentOption'.\\%
4671     Suggested actions:\\%
4672     * Make sure you haven't misspelled it\\%
4673     * Check in the babel manual that it's supported\\%
4674     * If supported and it's a language, you may\\%
4675     \space\space need in some distributions a separate\\%
4676     \space\space installation\\%
4677     * If installed, check there isn't an old\\%
4678     \space\space version of the required files in your system\\%
4679     * If it's an unsupported language, create it with\\%
4680     \string\babelprovide (see the manual)}
4681     {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4682     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4683     headfoot=, strings=, config=, hyphenmap=, or a language name.}
4684 \bbl@errmessage{config-not-found}
4685     {Local config file '\bbl@opt@config.cfg' not found.\\%
4686     Suggested actions:\\%
4687     * Make sure you haven't misspelled it in config=\\%
4688     * Check it exists and it's in the correct path}%
4689     {Perhaps you misspelled it.}
4690 \bbl@errmessage{late-after-babel}
4691     {Too late for \string\AfterBabelLanguage}%
4692     {Languages have been loaded, so I can do nothing}
4693 \bbl@errmessage{double-hyphens-class}
4694     {Double hyphens aren't allowed in \string\babelcharclass\\%
4695     because it's potentially ambiguous}%
4696     {See the manual for further info}

```

```

4697 \bbl@errmessage{unknown-interchar}
4698   {'#1' for '\language' cannot be enabled.\\%
4699   Maybe there is a typo}%
4700   {See the manual for further details.}
4701 \bbl@errmessage{unknown-interchar-b}
4702   {'#1' for '\language' cannot be disabled.\\%
4703   Maybe there is a typo}%
4704   {See the manual for further details.}
4705 \bbl@errmessage{charproperty-only-vertical}
4706   {\string\babelcharproperty\space can be used only in\\%
4707   vertical mode (preamble or between paragraphs)}%
4708   {See the manual for further info}
4709 \bbl@errmessage{unknown-char-property}
4710   {No property named '#2'. Allowed values are\\%
4711   direction (bc), mirror (bmg), and linebreak (lb)}%
4712   {See the manual for further info}
4713 \bbl@errmessage{bad-transform-option}
4714   {Bad option '#1' in a transform.\\%
4715   I'll ignore it but expect more errors}%
4716   {See the manual for further info.}
4717 \bbl@errmessage{font-conflict-transforms}
4718   {Transforms cannot be re-assigned to different\\%
4719   fonts. The conflict is in '\bbl@kv@label'.\\%
4720   Apply the same fonts or use a different label}%
4721   {See the manual for further details.}
4722 \bbl@errmessage{transform-not-available}
4723   {'#1' for '\language' cannot be enabled.\\%
4724   Maybe there is a typo or it's a font-dependent transform}%
4725   {See the manual for further details.}
4726 \bbl@errmessage{transform-not-available-b}
4727   {'#1' for '\language' cannot be disabled.\\%
4728   Maybe there is a typo or it's a font-dependent transform}%
4729   {See the manual for further details.}
4730 \bbl@errmessage{year-out-range}
4731   {Year out of range.\\%
4732   The allowed range is #1}%
4733   {See the manual for further details.}
4734 \bbl@errmessage{only-pdftex-lang}
4735   {The '#1' ldf style doesn't work with #2,\\%
4736   but you can use the ini locale instead.\\%
4737   Try adding 'provide=*' to the option list. You may\\%
4738   also want to set 'bidi=' to some value}%
4739   {See the manual for further details.}
4740 \bbl@errmessage{hyphenmins-args}
4741   {\string\babelhyphenmins\ accepts either the optional\\%
4742   argument or the star, but not both at the same time}%
4743   {See the manual for further details.}
4744 \bbl@errmessage{no-locale-for-meta}
4745   {There isn't currently a locale for the 'lang' requested\\%
4746   in the PDF metadata ('#1'). To fix it, you can\\%
4747   set explicitly a similar language (using the same\\%
4748   script) with the key main= when loading babel. If you\\%
4749   continue, I'll fallback to the 'nil' language, with\\%
4750   tag 'und' and script 'Latn', but expect a bad font\\%
4751   rendering with other scripts. You may also need set\\%
4752   explicitly captions and date, too}%
4753   {See the manual for further details.}
4754 </errors>
4755 <*patterns>

```

8. 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.

```
4756 <@Make sure ProvidesFile is defined>
4757 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4758 \xdef\bbl@format{\jobname}
4759 \def\bbl@version{<@version@>}
4760 \def\bbl@date{<@date@>}
4761 \ifx\AtBeginDocument\@undefined
4762   \def\@empty{}
4763 \fi
4764 <@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.

```
4765 \def\process@line#1#2 #3 #4 {%
4766   \ifx=#1%
4767     \process@synonym{#2}%
4768   \else
4769     \process@language{#1#2}{#3}{#4}%
4770   \fi
4771   \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.

```
4772 \toks@{}
4773 \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.

```
4774 \def\process@synonym#1{%
4775   \ifnum\last@language=\m@ne
4776     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4777   \else
4778     \expandafter\chardef\csname l@#1\endcsname\last@language
4779     \wlog{\string\l@#1=\string\language\the\last@language}%
4780     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4781       \csname\language\hyphenmins\endcsname
4782     \let\bbl@elt\relax
4783     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4784   \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`. $\text{\texttt{T\TeX}}$ does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<language>hyphenmins` macro. When no assignments were made we provide a default setting.

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

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

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

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

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

```

4785 \def\process@language#1#2#3{%
4786   \expandafter\addlanguage\csname l@#1\endcsname
4787   \expandafter\language\csname l@#1\endcsname
4788   \edef\language{#1}%
4789   \bbl@hook@everylanguage{#1}%
4790   % > luatex
4791   \bbl@get@enc#1::\@@@
4792   \begingroup
4793     \lefthyphenmin\m@ne
4794     \bbl@hook@loadpatterns{#2}%
4795     % > luatex
4796     \ifnum\lefthyphenmin=\m@ne
4797     \else
4798       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4799         \the\lefthyphenmin\the\righthyphenmin}%
4800     \fi
4801   \endgroup
4802   \def\bbl@tempa{#3}%
4803   \ifx\bbl@tempa\@empty\else
4804     \bbl@hook@loadexceptions{#3}%
4805     % > luatex
4806   \fi
4807   \let\bbl@elt\relax
4808   \edef\bbl@languages{%
4809     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4810   \ifnum\the\language=\z@
4811     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4812       \set@hyphenmins\tw@\thr@@\relax
4813     \else
4814       \expandafter\expandafter\expandafter\set@hyphenmins
4815       \csname #1hyphenmins\endcsname
4816     \fi
4817     \the\toks@
4818     \toks@{}%
4819   \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.

```

4820 \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.

```

4821 \def\bbl@hook@everylanguage#1{}
4822 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4823 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4824 \def\bbl@hook@loadkernel#1{%
4825   \def\addlanguage{\csname newlanguage\endcsname}%

```

```

4826 \def\adddialect##1##2{%
4827   \global\chardef##1##2\relax
4828   \log{\string##1 = a dialect from \string\language##2}}%
4829 \def\iflanguage##1{%
4830   \expandafter\ifx\csname l@##1\endcsname\relax
4831     \@nolanner{##1}%
4832   \else
4833     \ifnum\csname l@##1\endcsname=\language
4834       \expandafter\expandafter\expandafter\@firstoftwo
4835     \else
4836       \expandafter\expandafter\expandafter\@secondoftwo
4837     \fi
4838   \fi}%
4839 \def\providehyphenmins##1##2{%
4840   \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4841     \@namedef{##1hyphenmins}{##2}%
4842   \fi}%
4843 \def\set@hyphenmins##1##2{%
4844   \lefthyphenmin##1\relax
4845   \righthyphenmin##2\relax}%
4846 \def\selectlanguage{%
4847   \errhelp{Selecting a language requires a package supporting it}%
4848   \errmessage{No multilingual package has been loaded}}%
4849 \let\foreignlanguage\selectlanguage
4850 \let\otherlanguage\selectlanguage
4851 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4852 \def\bbl@usehooks##1##2{%
4853   \def\setlocale{%
4854     \errhelp{Find an armchair, sit down and wait}%
4855     \errmessage{(babel) Not yet available}}%
4856   \let\uselocale\setlocale
4857   \let\locale\setlocale
4858   \let\selectlocale\setlocale
4859   \let\localename\setlocale
4860   \let\textlocale\setlocale
4861   \let\textlanguage\setlocale
4862   \let\languagetext\setlocale}
4863 \begingroup
4864 \def\AddBabelHook#1#2{%
4865   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4866     \def\next{\toks1}%
4867   \else
4868     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4869   \fi
4870   \next}
4871 \ifx\directlua\@undefined
4872   \ifx\XeTeXinputencoding\@undefined\else
4873     \input xebabel.def
4874   \fi
4875 \else
4876   \input luababel.def
4877 \fi
4878 \openin1 = babel-\bbl@format.cfg
4879 \ifeof1
4880 \else
4881   \input babel-\bbl@format.cfg\relax
4882 \fi
4883 \closein1
4884 \endgroup
4885 \bbl@hook@loadkernel{switch.def}

```

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

```
4886 \openin1 = language.dat
```

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

```
4887 \def\language\language{english}%
4888 \ifeof1
4889   \message{I couldn't find the file language.dat,\space
4890           I will try the file hyphen.tex}
4891   \input hyphen.tex\relax
4892   \chardef\l@english\z@
4893 \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`.

```
4894 \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.

```
4895 \loop
4896   \endlinechar\m@ne
4897   \read1 to \bbl@line
4898   \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.

```
4899   \if T\ifeof1F\fi T\relax
4900   \ifx\bbl@line\empty\else
4901     \edef\bbl@line{\bbl@line\space\space\space}%
4902     \expandafter\process@line\bbl@line\relax
4903   \fi
4904 \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.

```
4905 \begingroup
4906   \def\bbl@elt#1#2#3#4{%
4907     \global\language=#2\relax
4908     \gdef\language\language{#1}%
4909     \def\bbl@elt##1##2##3##4{}}%
4910   \bbl@languages
4911 \endgroup
4912 \fi
4913 \closein1
```

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

```
4914 \if\the\toks@\else
4915   \errhelp{language.dat loads no language, only synonyms}
4916   \errmessage{Orphan language synonym}
4917 \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.

```
4918 \let\bbl@line\undefined
4919 \let\process@line\undefined
4920 \let\process@synonym\undefined
4921 \let\process@language\undefined
4922 \let\bbl@get@enc\undefined
4923 \let\bbl@hyph@enc\undefined
4924 \let\bbl@tempa\undefined
4925 \let\bbl@hook@loadkernel\undefined
4926 \let\bbl@hook@everylanguage\undefined
```

```

4927 \let\bbl@hook@loadpatterns\@undefined
4928 \let\bbl@hook@loadexceptions\@undefined
4929 </patterns>

```

Here the code for `iniTeX` ends.

9. luatex + xetex: common stuff

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

```

4930 <<*More package options>> ≡
4931 \chardef\bbl@bidimode\z@
4932 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4933 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4934 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4935 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4936 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4937 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4938 <</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`.

```

4939 <<*Font selection>> ≡
4940 \bbl@trace{Font handling with fontspec}
4941 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4942 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckestdfonts}
4943 \DisableBabelHook{babel-fontspec}
4944 \@onlypreamble\babelfont
4945 \ifx\NewDocumentCommand\@undefined\else % Not plain
4946   \NewDocumentCommand\babelfont{0}{m0}{m0}{}%
4947   \bbl@bblfont@o{#1}{#2}{#3,#5}{#4}}
4948 \fi
4949 \newcommand\bbl@bblfont@o[2][]{% 1=langs/scripts 2=fam
4950   \ifx\fontspec\@undefined
4951     \usepackage{fontspec}%
4952   \fi
4953   \EnableBabelHook{babel-fontspec}%
4954   \edef\bbl@tempa{#1}%
4955   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4956   \bbl@bblfont}
4957 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4958   \bbl@ifunset{\bbl@tempb family}%
4959   {\bbl@providfam{\bbl@tempb}}%
4960   {}%
4961   % For the default font, just in case:
4962   \bbl@ifunset{\bbl@sys\language}\bbl@provide@sys{\language}}{%
4963   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4964   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4965   \bbl@exp{%
4966     \let<\bbl@tempb dflt@\language>\<\bbl@tempb dflt@>%
4967     \bbl@font@set<\bbl@tempb dflt@\language>%
4968     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4969   {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4970     \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:

```

4971 \def\bbl@providfam#1{%
4972   \bbl@exp{%
4973     \newcommand\<#1default>{}% Just define it
4974     \bbl@add@list\bbl@font@fams{#1}%

```

```

4975 \\\NewHook{#1family}%
4976 \\\DeclareRobustCommand\<#1family>{%
4977   \\\not@math@alphabet\<#1family>\relax
4978   % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4979   \\\fontfamily\<#1default>%
4980   \\\UseHook{#1family}%
4981   \\\selectfont}%
4982 \\\DeclareTextFontCommand{\<text#1>}\<#1family>}}

```

The following macro is activated when the hook babel - fonts spec is enabled. But before, we define a macro for a warning, which sets a flag to avoid duplicate them.

```

4983 \def\bbbl@nostdfont#1{%
4984   \bbbl@once{nostdfam-\f@family}%
4985   {\bbbl@infowarn{The current font is not a babel standard family:\%
4986     #1%
4987     \fontname\font\\%
4988     There is nothing intrinsically wrong, and you can\\%,
4989     ignore this message altogether if you do not need\\%
4990     this font. If they are used in the document, be aware\\%
4991     'babel' will not set Script and Language for it, so\\%
4992     you may consider defining a new family with \string\babelfont.\\%
4993     See the manual for further details about \string\babelfont.
4994     Reported}}}%
4995   {}}%
4996 \gdef\bbbl@switchfont{%
4997   \bbbl@ifunset{bbbl@lsys@\language name}{\bbbl@provide@lsys{\language name}}}%
4998   \bbbl@exp{% e.g., Arabic -> arabic
4999     \lowercase{\edef\bbbl@tempa{\bbbl@c{l}{sname}}}}}%
5000 \bbbl@foreach\bbbl@font@fams{%
5001   \bbbl@ifunset{bbbl@##1dflt@\language name}% (1) language?
5002   {\bbbl@ifunset{bbbl@##1dflt*\bbbl@tempa}% (2) from script?
5003     {\bbbl@ifunset{bbbl@##1dflt@}% 2=F - (3) from generic?
5004       {}}% 123=F - nothing!
5005       {\bbbl@exp{% 3=T - from generic
5006         \global\let\<bbbl@##1dflt@\language name>%
5007         \<bbbl@##1dflt@>}}}%
5008       {\bbbl@exp{% 2=T - from script
5009         \global\let\<bbbl@##1dflt@\language name>%
5010         \<bbbl@##1dflt*\bbbl@tempa>}}}%
5011       {}}% 1=T - language, already defined
5012 \def\bbbl@tempa{\bbbl@nostdfont{}}}%
5013 \bbbl@foreach\bbbl@font@fams{% don't gather with prev for
5014   \bbbl@ifunset{bbbl@##1dflt@\language name}%
5015   {\bbbl@cs{famrst@##1}%
5016     \global\bbbl@csarg\let{famrst@##1}\relax}%
5017   {\bbbl@exp{% order is relevant.
5018     \\\bbbl@add\\originalTeX{%
5019       \\\bbbl@font@rst{\bbbl@c{l}{##1dflt}}}%
5020       \<##1default>\<##1family>{##1}}}%
5021     \\\bbbl@font@set\<bbbl@##1dflt@\language name>% the main part!
5022     \<##1default>\<##1family>}}}%
5023 \bbbl@ifrestoring{\bbbl@tempa}}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

5024 \ifx\f@family\undefined\else % if latex
5025   \ifcase\bbbl@engine % if pdftex
5026     \let\bbbl@ckeckstdfonts\relax
5027   \else
5028     \def\bbbl@ckeckstdfonts{%
5029       \begingroup
5030       \global\let\bbbl@ckeckstdfonts\relax
5031       \let\bbbl@tempa\empty
5032       \bbbl@foreach\bbbl@font@fams{%

```

```

5033 \bbl@ifunset{bbl@##1dflt}%
5034 {\@nameuse{##1family}}%
5035 \bbl@csarg\gdef{WFF@f@family}{}% Flag
5036 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \f@family\\}%
5037 \space\space\fontname\font\\}%
5038 \bbl@csarg\xdef{##1dflt@}{\f@family}%
5039 \expandafter\xdef\csname ##1default\endcsname{\f@family}%
5040 {}}%
5041 \ifx\bbl@tempa\empty\else
5042 \bbl@infowarn{The following font families will use the default\\%
5043 settings for all or some languages:\\%
5044 \bbl@tempa
5045 There is nothing intrinsically wrong with it, but\\%
5046 'babel' will no set Script and Language, which could\\%
5047 be relevant in some languages. If your document uses\\%
5048 these families, consider redefining them with \string\babelfont.\\%
5049 Reported}%
5050 \fi
5051 \endgroup}
5052 \fi
5053 \fi

```

Now the macros defining the font with fontspec.

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

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

```

5054 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
5055 \bbl@xin{<>}{#1}%
5056 \ifin@
5057 \bbl@exp{\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
5058 \fi
5059 \bbl@exp{%
5060 \def\\#2#1% e.g., \rmdefault{\bbl@rmdflt@lang}
5061 \\bbl@ifsamestring{#2}{\f@family}%
5062 {\\#3%
5063 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
5064 \let\\bbl@tempa\relax}%
5065 {}}}

```

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).

```

5066 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
5067 \let\bbl@tempa\bbl@mapselect
5068 \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
5069 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}}%
5070 \let\bbl@mapselect\relax
5071 \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
5072 \let#4\empty % Make sure \renewfontfamily is valid
5073 \bbl@set@renderer
5074 \bbl@exp{%
5075 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% e.g., '\rmfamily '
5076 <\keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
5077 {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
5078 <\keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
5079 {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%

```

```

5080   \\renewfontfamily\\#4%
5081   [\bbl@cl{lsys},% xetex removes unknown features :-(
5082   \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
5083   #2]}{#3}% i.e., \bbl@exp{..}{#3}
5084   \bbl@unset@renderer
5085   \begingroup
5086   #4%
5087   \xdef#1{\f@family}%      e.g., \bbl@rmdflt@lang{FreeSerif(0)}
5088   \endgroup
5089   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
5090   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
5091   \ifin@
5092   \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
5093   \fi
5094   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
5095   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
5096   \ifin@
5097   \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
5098   \fi
5099   \let#4\bbl@temp@fam
5100   \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
5101   \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.

```

5102 \def\bbl@font@rst#1#2#3#4{%
5103   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

5104 \def\bbl@font@fams{rm,sf,tt}
5105 <</Font selection>>

```

10. Hooks for XeTeX and LuaTeX

10.1. XeTeX

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

Now, the code.

```

5106 <*<xetex>
5107 \def\BabelStringsDefault{unicode}
5108 \let\xebbl@stop\relax
5109 \AddBabelHook{xetex}{encodedcommands}{%
5110   \def\bbl@tempa{#1}%
5111   \ifx\bbl@tempa\@empty
5112     \XeTeXinputencoding"bytes"%
5113   \else
5114     \XeTeXinputencoding"#1"%
5115   \fi
5116   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
5117 \AddBabelHook{xetex}{stopcommands}{%
5118   \xebbl@stop
5119   \let\xebbl@stop\relax}
5120 \def\bbl@input@classes{% Used in CJK intraspaces
5121   \input{load-unicode-xetex-classes.tex}%
5122   \let\bbl@input@classes\relax}
5123 \def\bbl@intraspace#1 #2 #3\@@{%
5124   \bbl@csarg\gdef{xeisp@{\languagename}%
5125     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
5126 \def\bbl@intrapenalty#1\@@{%
5127   \bbl@csarg\gdef{xeipn@{\languagename}%

```

```

5128     {\XeTeXlinebreakpenalty #1\relax}}
5129 \def\bbbl@provide@intraspace{%
5130   \bbbl@xin@{/s}{/\bbbl@cl{lnbrk}}}%
5131   \ifin@else\bbbl@xin@{/c}{/\bbbl@cl{lnbrk}}}\fi
5132   \ifin@
5133     \bbbl@ifunset{bbbl@intsp@{language\language}}{%
5134       {\expandafter\ifx\csname bbbl@intsp@{language\language}\endcsname\@empty\else
5135         \ifx\bbbl@KVP@intraspace\@nnil
5136           \bbbl@exp{%
5137             \\bbbl@intraspace\bbbl@cl{intsp}\@@}%
5138           \fi
5139           \ifx\bbbl@KVP@intrapenalty\@nnil
5140             \bbbl@intrapenalty0\@@
5141             \fi
5142             \fi
5143             \ifx\bbbl@KVP@intraspace\@nnil\else % We may override the ini
5144               \expandafter\bbbl@intraspace\bbbl@KVP@intraspace\@@
5145               \fi
5146               \ifx\bbbl@KVP@intrapenalty\@nnil\else
5147                 \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
5148                 \fi
5149                 \bbbl@exp{%
5150                   \\bbbl@add{<extras\language>{%
5151                     \XeTeXlinebreaklocale "\bbbl@cl{tbc}}}%
5152                     <bbbl@xeisp@{language}>%
5153                     <bbbl@xeipn@{language}>%
5154                     \\bbbl@toglobal{<extras\language>%
5155                     \\bbbl@add{<noextras\language>{%
5156                       \XeTeXlinebreaklocale ""}%
5157                       \\bbbl@toglobal{<noextras\language>%
5158                       \ifx\bbbl@ispace\undefined
5159                         \gdef\bbbl@ispace{\bbbl@cl{xeisp}}}%
5160                         \ifx\AtBeginDocument\@notprerr
5161                           \expandafter\@secondoftwo % to execute right now
5162                           \fi
5163                           \AtBeginDocument{\bbbl@patchfont{\bbbl@ispace}}}%
5164                           \fi}%
5165                           \fi}
5166 \ifx\DisableBabelHook\@undefined\endinput\fi
5167 \let\bbbl@set@renderer\relax
5168 \let\bbbl@unset@renderer\relax
5169 <@Font selection@>
5170 \def\bbbl@provide@extra#1{}

```

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

```

5171 \def\bbbl@xenohyph@{%
5172   \bbbl@ifset{bbbl@prehc@{language\language}}%
5173     {\ifnum\hyphenchar\font=\defaultthyphenchar
5174       \iffontchar\font\bbbl@cl{prehc}\relax
5175       \hyphenchar\font\bbbl@cl{prehc}\relax
5176       \else\iffontchar\font"200B
5177         \hyphenchar\font"200B
5178         \else
5179           \bbbl@warning
5180             {Neither 0 nor ZERO WIDTH SPACE are available\\%
5181             in the current font, and therefore the hyphen\\%
5182             will be printed. Try changing the fontspec's\\%
5183             'HyphenChar' to another value, but be aware\\%
5184             this setting is not safe (see the manual).\\%
5185             Reported}%
5186           \hyphenchar\font\defaultthyphenchar
5187           \fi\fi
5188           \fi}%

```

5189 {\hyphenchar\font\defaultthyphenchar}}

10.2. Support for interchar

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

```
5190 \ifnum\Xe@alloc@intercharclass<\thr@@
5191   \Xe@alloc@intercharclass\thr@@
5192 \fi
5193 \chardef\bbl@xe@class@default@=\z@
5194 \chardef\bbl@xe@class@cjkideogram@=\@ne
5195 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
5196 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
5197 \chardef\bbl@xe@class@boundary@=4095
5198 \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.

```
5199 \AddBabelHook{babel-interchar}{beforeextras}{%
5200   \@nameuse{bbl@xechars@\language@}}
5201 \DisableBabelHook{babel-interchar}
5202 \protected\def\bbl@charclass#1{%
5203   \ifnum\count@<\z@
5204     \count@-\count@
5205     \loop
5206       \bbl@exp{%
5207         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5208         \XeTeXcharclass\count@ \bbl@tempc
5209         \ifnum\count@<`#1\relax
5210           \advance\count@\@ne
5211         \repeat
5212   \else
5213     \babel@savevariable{\XeTeXcharclass`#1}%
5214     \XeTeXcharclass`#1 \bbl@tempc
5215   \fi
5216   \count@`#1\relax}
```

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

```
5217 \newcommand\bbl@ifinterchar[1]{%
5218   \let\bbl@tempa\@gobble           % Assume to ignore
5219   \edef\bbl@tempb{\zap@space#1 \@empty}%
5220   \ifx\bbl@KVP@interchar\@nnil\else
5221     \bbl@replace\bbl@KVP@interchar{ }{,}%
5222     \bbl@foreach\bbl@tempb{%
5223       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5224     \ifin@
5225       \let\bbl@tempa\@firstofone
5226     \fi}%
5227 \fi
5228 \bbl@tempa}
5229 \newcommand\IfBabelIntercharT[2]{%
5230   \bbl@carg\bbl@add{bbl@icsave\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5231 \newcommand\babelcharclass[3]{%
5232   \EnableBabelHook{babel-interchar}%
5233   \bbl@carg\newXeTeXintercharclass{xe@class@#2@#1}%
5234   \def\bbl@tempb##1{%
```

```

5235 \ifx##1\@empty\else
5236 \ifx##1-%
5237 \bbl@upto
5238 \else
5239 \bbl@class{%-
5240 \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5241 \fi
5242 \expandafter\bbl@tempb
5243 \fi}%
5244 \bbl@ifunset{\bbl@xechars@#1}%
5245 {\toks@{%-
5246 \babel@savevariable\XeTeXinterchartokenstate
5247 \XeTeXinterchartokenstate\@ne
5248 }}%
5249 {\toks@\expandafter\expandafter\expandafter{%-
5250 \csname bbl@xechars@#1\endcsname}}%
5251 \bbl@csarg\edef{\xechars@#1}{%-
5252 \the\toks@
5253 \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
5254 \bbl@tempb#3\@empty}}
5255 \protected\def\bbl@usingxeclasse#1{\count@ \z@ \let\bbl@tempc#1}
5256 \protected\def\bbl@upto{%-
5257 \ifnum\count@>\z@
5258 \advance\count@\@ne
5259 \count@-\count@
5260 \else\ifnum\count@=\z@
5261 \bbl@class{-}%
5262 \else
5263 \bbl@error{double-hyphens-class}{\count@}{\count@}%
5264 \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>`.

```

5265 \def\bbl@ignoreinterchar{%-
5266 \ifnum\language=\l@nohyphenation
5267 \expandafter\@gobble
5268 \else
5269 \expandafter\@firstofone
5270 \fi}
5271 \newcommand\babelinterchar[5][{}]{%-
5272 \let\bbl@kv@label\@empty
5273 \bbl@forkv{#1}{\bbl@csarg\edef{\kv@##1}{##2}}%
5274 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5275 {\bbl@ignoreinterchar{#5}}%
5276 \bbl@csarg\let{\ic@\bbl@kv@label @#2}\@firstofone
5277 \bbl@expf{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%-
5278 \bbl@expf{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%-
5279 \XeTeXinterchartoks
5280 \@nameuse{\bbl@xeclasse@\bbl@tempa @#2}%
5281 \bbl@ifunset{\bbl@xeclasse@\bbl@tempa @#2}{\bbl@tempa @#2}%
5282 \@nameuse{\bbl@xeclasse@\bbl@tempb @#2}%
5283 \bbl@ifunset{\bbl@xeclasse@\bbl@tempb @#2}{\bbl@tempb @#2}%
5284 = \expandafter{%-
5285 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5286 \csname\zap@space bbl@xeinter@\bbl@kv@label
5287 @#3@#4@#2 \@empty\endcsname}}}%
5288 \DeclareRobustCommand\enablelocaleinterchar[1]{%-
5289 \bbl@ifunset{\bbl@ic@#1@language}%
5290 {\bbl@error{unknown-interchar}{#1}{\count@}}%
5291 {\bbl@csarg\let{\ic@#1@language}\@firstofone}}
5292 \DeclareRobustCommand\disablelocaleinterchar[1]{%-
5293 \bbl@ifunset{\bbl@ic@#1@language}%

```

```

5294     {\bbl@error{unknown-interchar-b}{#1}{}}}%
5295     {\bbl@csarg\let{ic@#1@\language\@gobble}}
5296 \</xetex>

```

10.3. Layout

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

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the 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 pdftex and xetex.

```

5297 <*xetex | texxet>
5298 \providecommand\bbl@provide@intraspace{}
5299 \bbl@trace{Redefinitions for bidi layout}

    Finish here if there in no layout.

5300 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5301   \ifx\bbl@opt@layout\@undefined\else % Plain
5302     \IfBabelLayout{nopars}
5303     {}
5304     {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5305   \fi
5306 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5307 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5308 \ifnum\bbl@bidimode>\z@
5309 \IfBabelLayout{pars}
5310   {\def\@hangfrom#1{%
5311     \setbox\@tempboxa\hbox{#{1}}%
5312     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5313     \noindent\box\@tempboxa}
5314   \def\raggedright{%
5315     \let\\\@centercr
5316     \bbl@startskip\z@skip
5317     \@rightskip\@flushglue
5318     \bbl@endskip\@rightskip
5319     \parindent\z@
5320     \parfillskip\bbl@startskip}
5321   \def\raggedleft{%
5322     \let\\\@centercr
5323     \bbl@startskip\@flushglue
5324     \bbl@endskip\z@skip
5325     \parindent\z@
5326     \parfillskip\bbl@endskip}}
5327   {}
5328 \fi
5329 \IfBabelLayout{lists}
5330   {\bbl@sreplace\list
5331     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5332     \def\bbl@listleftmargin{%
5333       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5334     \ifcase\bbl@engine
5335       \def\labelenumii{}\theenumii{}% pdfTeX doesn't reverse ()
5336       \def\p@enumiii{\p@enumii}\theenumii{}%
5337     \fi
5338     \bbl@sreplace\@verbatim
5339     {\leftskip\@totalleftmargin}%
5340     {\bbl@startskip\textwidth
5341       \advance\bbl@startskip-\linewidth}%
5342     \bbl@sreplace\@verbatim
5343     {\rightskip\z@skip}%
5344     {\bbl@endskip\z@skip}}%

```

```

5345 {}
5346 \IfBabelLayout{contents}
5347 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5348 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5349 {}
5350 \IfBabelLayout{columns}
5351 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5352 \def\bbl@outputbox#1{%
5353 \hb@xt@\textwidth{%
5354 \hskip\columnwidth
5355 \hfil
5356 {\normalcolor\vrule \@width\columnseprule}%
5357 \hfil
5358 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5359 \hskip-\textwidth
5360 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5361 \hskip\columnsep
5362 \hskip\columnwidth}}}%
5363 {}

```

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.

```

5364 \IfBabelLayout{counters*}%
5365 {\bbl@add\bbl@opt@layout{.counters.}%
5366 \AddToHook{shipout/before}{%
5367 \let\bbl@tempa\babelsublr
5368 \let\babelsublr\@firstofone
5369 \let\bbl@save@thepage\thepage
5370 \protected@edef\thepage{\thepage}%
5371 \let\babelsublr\bbl@tempa}%
5372 \AddToHook{shipout/after}{%
5373 \let\thepage\bbl@save@thepage}}{}
5374 \IfBabelLayout{counters}%
5375 {\let\bbl@latinarabic=\@arabic
5376 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5377 \let\bbl@asciroman=\@roman
5378 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5379 \let\bbl@asciiRoman=\@Roman
5380 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5381 \fi % end if layout
5382 </xetex | texxet>

```

10.4. 8-bit TeX

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

```

5383 <*texxet>
5384 \def\bbl@provide@extra#1{%
5385 % == auto-select encoding ==
5386 \ifx\bbl@encoding@select@off\@empty\else
5387 \bbl@ifunset{\bbl@encoding@#1}%
5388 {\def\@elt##1{,##1,}%
5389 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5390 \count@\z@
5391 \bbl@foreach\bbl@tempe{%
5392 \def\bbl@tempd{##1}% Save last declared
5393 \advance\count@\@ne}%
5394 \ifnum\count@>\@ne % (1)
5395 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5396 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5397 \bbl@replace\bbl@tempa{ },}%
5398 \global\bbl@csarg\let{encoding@#1}\@empty
5399 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%

```

```

5400     \ifin\else % if main encoding included in ini, do nothing
5401     \let\bbl@tempb\relax
5402     \bbl@foreach\bbl@tempa{%
5403     \ifx\bbl@tempb\relax
5404     \bbl@xin@{,##1,}{,\bbl@tempe,}%
5405     \ifin\def\bbl@tempb{##1}\fi
5406     \fi}%
5407     \ifx\bbl@tempb\relax\else
5408     \bbl@exp{%
5409     \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5410     \gdef\<bbl@encoding@#1>{%
5411     \\babel@save\\f@encoding
5412     \\bbl@add\\originalTeX{\\selectfont}%
5413     \\fontencoding{\bbl@tempb}%
5414     \\selectfont}}%
5415     \fi
5416     \fi
5417     \fi}%
5418     }%
5419     \fi}
5420 </texxet>

```

10.5. LuaTeX

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

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

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

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

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

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

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

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

```

5421 <*\luatex>
5422 \directlua{ Babel = Babel or {} } % DL2
5423 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5424 \bbl@trace{Read language.dat}
5425 \ifx\bbl@readstream\undefined
5426 \csname newread\endcsname\bbl@readstream
5427 \fi

```

```

5428 \begingroup
5429 \toks@{}
5430 \count@ \z@ % 0=start, 1=0th, 2=normal
5431 \def\bbl@process@line#1#2 #3 #4 {%
5432 \ifx=#1%
5433 \bbl@process@synonym{#2}%
5434 \else
5435 \bbl@process@language{#1#2}{#3}{#4}%
5436 \fi
5437 \ignorespaces}
5438 \def\bbl@manylang{%
5439 \ifnum\bbl@last>\@ne
5440 \bbl@info{Non-standard hyphenation setup}%
5441 \fi
5442 \let\bbl@manylang\relax}
5443 \def\bbl@process@language#1#2#3{%
5444 \ifcase\count@
5445 \@@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5446 \or
5447 \count@\tw@
5448 \fi
5449 \ifnum\count@=\tw@
5450 \expandafter\addlanguage\csname l@#1\endcsname
5451 \language\allocationnumber
5452 \chardef\bbl@last\allocationnumber
5453 \bbl@manylang
5454 \let\bbl@elt\relax
5455 \xdef\bbl@languages{%
5456 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5457 \fi
5458 \the\toks@
5459 \toks@{}}
5460 \def\bbl@process@synonym@aux#1#2{%
5461 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5462 \let\bbl@elt\relax
5463 \xdef\bbl@languages{%
5464 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5465 \def\bbl@process@synonym#1{%
5466 \ifcase\count@
5467 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5468 \or
5469 \@@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5470 \else
5471 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5472 \fi}
5473 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
5474 \chardef\l@english\z@
5475 \chardef\l@USenglish\z@
5476 \chardef\bbl@last\z@
5477 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5478 \gdef\bbl@languages{%
5479 \bbl@elt{english}{0}{hyphen.tex}}%
5480 \bbl@elt{USenglish}{0}{}%
5481 \else
5482 \global\let\bbl@languages@format\bbl@languages
5483 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5484 \ifnum#2>\z@\else
5485 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5486 \fi}%
5487 \xdef\bbl@languages{\bbl@languages}%
5488 \fi
5489 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
5490 \bbl@languages

```

```

5491 \openin\bbl@readstream=language.dat
5492 \ifeof\bbl@readstream
5493   \bbl@warning{I couldn't find language.dat. No additional\\%
5494               patterns loaded. Reported}%
5495 \else
5496   \loop
5497     \endlinechar\m@ne
5498     \read\bbl@readstream to \bbl@line
5499     \endlinechar`\^^M
5500     \if T\ifeof\bbl@readstream F\fi T\relax
5501     \ifx\bbl@line\@empty\else
5502       \edef\bbl@line{\bbl@line\space\space\space}%
5503       \expandafter\bbl@process@line\bbl@line\relax
5504     \fi
5505   \repeat
5506 \fi
5507 \closein\bbl@readstream
5508 \endgroup
5509 \bbl@trace{Macros for reading patterns files}
5510 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5511 \ifx\babelcatcodetablenum\@undefined
5512   \ifx\newcatcodetable\@undefined
5513     \def\babelcatcodetablenum{5211}
5514     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5515   \else
5516     \newcatcodetable\babelcatcodetablenum
5517     \newcatcodetable\bbl@pattcodes
5518   \fi
5519 \else
5520   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5521 \fi
5522 \def\bbl@luapatterns#1#2{%
5523   \bbl@get@enc#1:.\@@@
5524   \setbox\z@\hbox\bgroup
5525     \begin{group}
5526       \savecatcodetable\babelcatcodetablenum\relax
5527       \initcatcodetable\bbl@pattcodes\relax
5528       \catcodetable\bbl@pattcodes\relax
5529       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5530       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
5531       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5532       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5533       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5534       \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5535       \input #1\relax
5536     \catcodetable\babelcatcodetablenum\relax
5537   \endgroup
5538   \def\bbl@tempa{#2}%
5539   \ifx\bbl@tempa\@empty\else
5540     \input #2\relax
5541   \fi
5542 \egroup}%
5543 \def\bbl@patterns@lua#1{%
5544   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5545     \csname l@#1\endcsname
5546     \edef\bbl@tempa{#1}%
5547   \else
5548     \csname l@#1:\f@encoding\endcsname
5549     \edef\bbl@tempa{#1:\f@encoding}%
5550   \fi\relax
5551   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5552   \@ifundefined{bbl@hyphendata@the\language}%
5553     {\def\bbl@elt##1##2##3##4{%

```

```

5554 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5555 \def\bbl@tempb{##3}%
5556 \ifx\bbl@tempb\empty\else % if not a synonymous
5557 \def\bbl@tempc{{##3}{##4}}%
5558 \fi
5559 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5560 \fi}%
5561 \bbl@languages
5562 \@ifundefined{bbl@hyphendata@the\language}%
5563 {\bbl@info{No hyphenation patterns were set for\%
5564 language '\bbl@tempa'. Reported}}%
5565 {\expandafter\expandafter\expandafter\bbl@luapatterns
5566 \csname bbl@hyphendata@the\language\endcsname}}}%
5567 \endinput\fi

```

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

```

5568 \ifx\DisableBabelHook\@undefined
5569 \AddBabelHook{luatex}{everylanguage}{%
5570 \def\process@language##1##2##3{%
5571 \def\process@line####1####2 ####3 ####4 {}}
5572 \AddBabelHook{luatex}{loadpatterns}{%
5573 \input #1\relax
5574 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5575 {{#1}}}%
5576 \AddBabelHook{luatex}{loadexceptions}{%
5577 \input #1\relax
5578 \def\bbl@tempb##1##2{{##1}{##2}}%
5579 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5580 {\expandafter\expandafter\expandafter\bbl@tempb
5581 \csname bbl@hyphendata@the\language\endcsname}}
5582 \endinput\fi

```

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

```

5583 \begin{group}
5584 \catcode`\%=12
5585 \catcode`\'=12
5586 \catcode`\%=12
5587 \catcode`\:=12
5588 \directlua{
5589 Babel.locale_props = Babel.locale_props or {}
5590 function Babel.lua_error(e, a)
5591 tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5592 e .. '}' .. (a or '') .. '}{'}])
5593 end
5594
5595 function Babel.bytes(line)
5596 return line:gsub(".",
5597 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5598 end
5599
5600 function Babel.priority_in_callback(name,description)
5601 for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5602 if v == description then return i end
5603 end
5604 return false
5605 end
5606
5607 function Babel.begin_process_input()
5608 if luatexbase and luatexbase.add_to_callback then
5609 luatexbase.add_to_callback('process_input_buffer',
5610 Babel.bytes, 'Babel.bytes')
5611 else
5612 Babel.callback = callback.find('process_input_buffer')

```

```

5613     callback.register('process_input_buffer',Babel.bytes)
5614 end
5615 end
5616 function Babel.end_process_input ()
5617     if luatexbase and luatexbase.remove_from_callback then
5618         luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5619     else
5620         callback.register('process_input_buffer',Babel.callback)
5621     end
5622 end
5623
5624 function Babel.str_to_nodes(fn, matches, base)
5625     local n, head, last
5626     if fn == nil then return nil end
5627     for s in string.utfvalues(fn(matches)) do
5628         if base.id == 7 then
5629             base = base.replace
5630         end
5631         n = node.copy(base)
5632         n.char = s
5633         if not head then
5634             head = n
5635         else
5636             last.next = n
5637         end
5638         last = n
5639     end
5640     return head
5641 end
5642
5643 Babel.linebreaking = Babel.linebreaking or {}
5644 Babel.linebreaking.before = {}
5645 Babel.linebreaking.after = {}
5646 Babel.locale = {}
5647 function Babel.linebreaking.add_before(func, pos)
5648     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5649     if pos == nil then
5650         table.insert(Babel.linebreaking.before, func)
5651     else
5652         table.insert(Babel.linebreaking.before, pos, func)
5653     end
5654 end
5655 function Babel.linebreaking.add_after(func)
5656     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5657     table.insert(Babel.linebreaking.after, func)
5658 end
5659
5660 function Babel.addpatterns(pp, lg)
5661     local lg = lang.new(lg)
5662     local pats = lang.patterns(lg) or ''
5663     lang.clear_patterns(lg)
5664     for p in pp:gmatch('[^%s]+') do
5665         ss = ''
5666         for i in string.utfcharacters(p:gsub('%d', '')) do
5667             ss = ss .. '%d?' .. i
5668         end
5669         ss = ss:gsub('^%d%?%.','%.') .. '%d?'
5670         ss = ss:gsub('%.%d%?$', '%%.')
5671         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5672         if n == 0 then
5673             tex.sprint(
5674                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5675                 .. p .. [[]])

```

```

5676     pats = pats .. ' ' .. p
5677 else
5678     tex.sprint(
5679         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]
5680         .. p .. [{}]])
5681     end
5682 end
5683 lang.patterns(lg, pats)
5684 end
5685
5686 Babel.characters = Babel.characters or {}
5687 Babel.ranges = Babel.ranges or {}
5688 function Babel.hlist_has_bidi(head)
5689     local has_bidi = false
5690     local ranges = Babel.ranges
5691     for item in node.traverse(head) do
5692         if item.id == node.id'glyph' then
5693             local itemchar = item.char
5694             local chardata = Babel.characters[itemchar]
5695             local dir = chardata and chardata.d or nil
5696             if not dir then
5697                 for nn, et in ipairs(ranges) do
5698                     if itemchar < et[1] then
5699                         break
5700                     elseif itemchar <= et[2] then
5701                         dir = et[3]
5702                         break
5703                     end
5704                 end
5705             end
5706             if dir and (dir == 'al' or dir == 'r') then
5707                 has_bidi = true
5708             end
5709         end
5710     end
5711     return has_bidi
5712 end
5713 function Babel.set_chranges_b (script, chrng)
5714     if chrng == '' then return end
5715     texio.write('Replacing ' .. script .. ' script ranges')
5716     Babel.script_blocks[script] = {}
5717     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.)%s') do
5718         table.insert(
5719             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5720     end
5721 end
5722
5723 function Babel.discard_sublr(str)
5724     if str:find( [[\string\indexentry]] ) and
5725        str:find( [[\string\babelsublr]] ) then
5726         str = str:gsub( [[\string\babelsublr%s*{%b{}}]],
5727             function(m) return m:sub(2,-2) end )
5728     end
5729     return str
5730 end
5731 }
5732 \endgroup
5733 \ifx\newattribute\undefined\else % Test for plain
5734     \newattribute\bbl@attr@locale % DL4
5735     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5736     \AddBabelHook{luatex}{beforeextras}{%
5737         \setattribute\bbl@attr@locale\localeid}
5738 \fi

```

```

5739 %
5740 \def\BabelStringsDefault{unicode}
5741 \let\luabbl@stop\relax
5742 \AddBabelHook{luatex}{encodedcommands}{%
5743   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5744   \ifx\bbl@tempa\bbl@tempb\else
5745     \directlua{Babel.begin_process_input()}%
5746     \def\luabbl@stop{%
5747       \directlua{Babel.end_process_input()}}%
5748   \fi}%
5749 \AddBabelHook{luatex}{stopcommands}{%
5750   \luabbl@stop
5751   \let\luabbl@stop\relax}
5752 %
5753 \AddBabelHook{luatex}{patterns}{%
5754   \@ifundefined{bbl@hyphendata@the\language}%
5755   {\def\bbl@elt##1##2##3##4{%
5756     \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5757     \def\bbl@tempb{##3}%
5758     \ifx\bbl@tempb\@empty\else % if not a synonymous
5759       \def\bbl@tempc{##3}{##4}%
5760       \fi
5761       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5762     \fi}%
5763   \bbl@languages
5764   \@ifundefined{bbl@hyphendata@the\language}%
5765   {\bbl@info{No hyphenation patterns were set for\%
5766     language '#2'. Reported}}%
5767   {\expandafter\expandafter\expandafter\bbl@luapatterns
5768     \csname bbl@hyphendata@the\language\endcsname}}}%
5769 \@ifundefined{bbl@patterns@}{%
5770   \begingroup
5771   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5772   \ifin@else
5773     \ifx\bbl@patterns@\@empty\else
5774       \directlua{ Babel.addpatterns(
5775         [[\bbl@patterns@]], \number\language) }%
5776     \fi
5777     \@ifundefined{bbl@patterns@#1}%
5778     \@empty
5779     {\directlua{ Babel.addpatterns(
5780       [[\space\csname bbl@patterns@#1\endcsname]],
5781       \number\language) }}%
5782     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,%
5783     \fi
5784   \endgroup}%
5785   \bbl@exp{%
5786     \bbl@ifunset{bbl@prehc@languagename}{}%
5787     {\bbl@ifblank{\bbl@cs{prehc@languagename}}}%
5788     {\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.

```

5789 \@onlypreamble\babelpatterns
5790 \AtEndOfPackage{%
5791   \newcommand\babelpatterns[2][\@empty]{%
5792     \ifx\bbl@patterns@\relax
5793       \let\bbl@patterns@\@empty
5794     \fi
5795     \ifx\bbl@pttnlist@\@empty\else
5796       \bbl@warning{%
5797         You must not intermingle \string\selectlanguage\space and\%

```

```

5798     \string\babelpatterns\space or some patterns will not\\%
5799     be taken into account. Reported}%
5800 \fi
5801 \ifx\@empty#1%
5802     \protected@edef\bbbl@patterns@\bbbl@patterns@\space#2}%
5803 \else
5804     \edef\bbbl@tempb{\zap@space#1 \@empty}%
5805     \bbbl@for\bbbl@tempa\bbbl@tempb{%
5806         \bbbl@fixname\bbbl@tempa
5807         \bbbl@iflanguage\bbbl@tempa{%
5808             \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
5809                 \@ifundefined{bbbl@patterns@\bbbl@tempa}%
5810                 \@empty
5811                 {\csname bbl@patterns@\bbbl@tempa\endcsname\space}%
5812                 #2}}}%
5813 \fi}}

```

10.6. Southeast Asian scripts

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

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

```

5814 \def\bbbl@intraspace#1 #2 #3\@{
5815     \directlua{
5816         Babel.intraspaces = Babel.intraspaces or {}
5817         Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5818             {b = #1, p = #2, m = #3}
5819         Babel.locale_props[\the\localeid].intraspace = %
5820             {b = #1, p = #2, m = #3}
5821     }}
5822 \def\bbbl@intrapenalty#1\@{
5823     \directlua{
5824         Babel.intrapenalties = Babel.intrapenalties or {}
5825         Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5826         Babel.locale_props[\the\localeid].intrapenalty = #1
5827     }}
5828 \begingroup
5829 \catcode`\%=12
5830 \catcode`\&=14
5831 \catcode`\'=12
5832 \catcode`\-=12
5833 \gdef\bbbl@seaintraspace{&
5834     \let\bbbl@seaintraspace\relax
5835     \directlua{
5836         Babel.sea_enabled = true
5837         Babel.sea_ranges = Babel.sea_ranges or {}
5838         function Babel.set_chranges (script, chrng)
5839             local c = 0
5840             for s, e in string.gmatch(chrng..' ', '(.-%.%.(.-%s)') do
5841                 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5842                 c = c + 1
5843             end
5844         end
5845         function Babel.sea_disc_to_space (head)
5846             local sea_ranges = Babel.sea_ranges
5847             local last_char = nil
5848             local quad = 655360 & 10 pt = 655360 = 10 * 65536
5849             for item in node.traverse(head) do
5850                 local i = item.id
5851                 if i == node.id'glyph' then
5852                     last_char = item
5853                 elseif i == 7 and item.subtype == 3 and last_char

```

```

5854         and last_char.char > 0x0C99 then
5855     quad = font.getfont(last_char.font).size
5856     for lg, rg in pairs(sea_ranges) do
5857         if last_char.char > rg[1] and last_char.char < rg[2] then
5858             lg = lg:sub(1, 4)  &% Remove trailing number of, e.g., Cyril
5859             local intraspace = Babel.intraspaces[lg]
5860             local intrapenalty = Babel.intrapenalties[lg]
5861             local n
5862             if intrapenalty ~= 0 then
5863                 n = node.new(14, 0)    &% penalty
5864                 n.penalty = intrapenalty
5865                 node.insert_before(head, item, n)
5866             end
5867             n = node.new(12, 13)      &% (glue, spaceskip)
5868             node.setglue(n, intraspace.b * quad,
5869                           intraspace.p * quad,
5870                           intraspace.m * quad)
5871             node.insert_before(head, item, n)
5872             node.remove(head, item)
5873         end
5874     end
5875 end
5876 end
5877 end
5878 }&
5879 \bbl@luahyphenate}

```

10.7. CJK line breaking

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

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

```

5880 \catcode`\%=14
5881 \gdef\bbl@cjkintraspaces{%
5882   \let\bbl@cjkintraspaces\relax
5883   \directlua{
5884     require('babel-data-cjk.lua')
5885     Babel.cjk_enabled = true
5886     function Babel.cjk_linebreak(head)
5887         local GLYPH = node.id'glyph'
5888         local last_char = nil
5889         local quad = 655360      % 10 pt = 655360 = 10 * 65536
5890         local last_class = nil
5891         local last_lang = nil
5892         for item in node.traverse(head) do
5893             if item.id == GLYPH then
5894                 local lang = item.lang
5895                 local LOCALE = node.get_attribute(item,
5896                     Babel.attr_locale)
5897                 local props = Babel.locale_props[LOCALE] or {}
5898                 local class = Babel.cjk_class[item.char].c
5899                 if props.cjk_quotes and props.cjk_quotes[item.char] then
5900                     class = props.cjk_quotes[item.char]
5901                 end
5902                 if class == 'cp' then class = 'cl' % }) as CL
5903                 elseif class == 'id' then class = 'I'
5904                 elseif class == 'cj' then class = 'I' % loose
5905                 end
5906                 local br = 0
5907                 if class and last_class and Babel.cjk_breaks[last_class][class] then

```

```

5908         br = Babel.cjk_breaks[last_class][class]
5909     end
5910     if br == 1 and props.linebreak == 'c' and
5911         lang ~= \the\l@nohyphenation\space and
5912         last_lang ~= \the\l@nohyphenation then
5913         local intrapenalty = props.intrapenalty
5914         if intrapenalty ~= 0 then
5915             local n = node.new(14, 0)      % penalty
5916             n.penalty = intrapenalty
5917             node.insert_before(head, item, n)
5918         end
5919         local intraspace = props.intraspace
5920         local n = node.new(12, 13)      % (glue, spaceskip)
5921         node.setglue(n, intraspace.b * quad,
5922             intraspace.p * quad,
5923             intraspace.m * quad)
5924         node.insert_before(head, item, n)
5925     end
5926     if font.getfont(item.font) then
5927         quad = font.getfont(item.font).size
5928     end
5929     last_class = class
5930     last_lang = lang
5931     else % if penalty, glue or anything else
5932         last_class = nil
5933     end
5934 end
5935 lang.hyphenate(head)
5936 end
5937 }%
5938 \bbl@luahyphenate}
5939 \gdef\bbl@luahyphenate{%
5940 \let\bbl@luahyphenate\relax
5941 \directlua{
5942     luatexbase.add_to_callback('hyphenate',
5943     function (head, tail)
5944         if Babel.linebreaking.before then
5945             for k, func in ipairs(Babel.linebreaking.before) do
5946                 func(head)
5947             end
5948         end
5949         lang.hyphenate(head)
5950         if Babel.cjk_enabled then
5951             Babel.cjk_linebreak(head)
5952         end
5953         if Babel.linebreaking.after then
5954             for k, func in ipairs(Babel.linebreaking.after) do
5955                 func(head)
5956             end
5957         end
5958         if Babel.set_hboxed then
5959             Babel.set_hboxed(head)
5960         end
5961         if Babel.sea_enabled then
5962             Babel.sea_disc_to_space(head)
5963         end
5964     end,
5965     'Babel.hyphenate')
5966 }}
5967 \endgroup
5968 %
5969 \def\bbl@provide@intraspace{%
5970 \bbl@ifunset\bbl@intsp@{language}{}}%

```

```

5971 {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
5972 \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}}%
5973 \ifin@ % cjk
5974 \bbl@cjkintraspacespace
5975 \directlua{
5976 Babel.locale_props = Babel.locale_props or {}
5977 Babel.locale_props[\the\localeid].linebreak = 'c'
5978 }%
5979 \bbl@exp{\bbl@intraspacespace\bbl@cl{intsp}}\bbl@cl{}%
5980 \ifx\bbl@KVP@intrapenalty\@nnil
5981 \bbl@intrapenalty0\@
5982 \fi
5983 \else % sea
5984 \bbl@seaintraspacespace
5985 \bbl@exp{\bbl@intraspacespace\bbl@cl{intsp}}\bbl@cl{}%
5986 \directlua{
5987 Babel.sea_ranges = Babel.sea_ranges or {}
5988 Babel.set_chranges('\bbl@cl{sbcpr}',
5989 '\bbl@cl{chrng}')
5990 }%
5991 \ifx\bbl@KVP@intrapenalty\@nnil
5992 \bbl@intrapenalty0\@
5993 \fi
5994 \fi
5995 \fi
5996 \ifx\bbl@KVP@intrapenalty\@nnil\else
5997 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5998 \fi}}

```

10.8. Arabic justification

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

```

5999 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
6000 \def\bblar@chars{%
6001 0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
6002 0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
6003 0640,0641,0642,0643,0644,0645,0646,0647,0649}
6004 \def\bblar@elongated{%
6005 0626,0628,062A,062B,0633,0634,0635,0636,063B,%
6006 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
6007 0649,064A}
6008 \begingroup
6009 \catcode`\_ =11 \catcode`\:=11
6010 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
6011 \endgroup
6012 \gdef\bbl@arabicjust{%
6013 \let\bbl@arabicjust\relax
6014 \newattribute\bblar@kashida
6015 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
6016 \bblar@kashida=\z@
6017 \bbl@patchfont{\bbl@parsejalt}}%
6018 \directlua{
6019 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
6020 Babel.arabic.elong_map[\the\localeid] = {}
6021 luatexbase.add_to_callback('post_linebreak_filter',
6022 Babel.arabic.justify, 'Babel.arabic.justify')
6023 luatexbase.add_to_callback('hpack_filter',
6024 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
6025 }}%

```

Save both node lists to make replacement.

```

6026 \def\bblar@fetchjalt#1#2#3#4{%

```

```

6027 \bbl@exp{\bbl@foreach{#1}}{%
6028   \bbl@ifunset{bblar@JE@##1}%
6029     {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
6030     {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
6031   \directlua{%
6032     local last = nil
6033     for item in node.traverse(tex.box[0].head) do
6034       if item.id == node.id'glyph' and item.char > 0x600 and
6035         not (item.char == 0x200D) then
6036         last = item
6037       end
6038     end
6039     Babel.arabic.#3['##1#4'] = last.char
6040   }}

```

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

```

6041 \gdef\bbl@parsejalt{%
6042   \ifx\addfontfeature\undefined\else
6043     \bbl@xin{/e}{/\bbl@cl{\lnbrk}}%
6044     \ifin@
6045       \directlua{%
6046         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
6047           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
6048           tex.print([[string\cswb\space bbl@parsejalti\endcswb]])
6049         end
6050       }%
6051     \fi
6052   \fi}
6053 \gdef\bbl@parsejalti{%
6054   \begingroup
6055     \let\bbl@parsejalt\relax % To avoid infinite loop
6056     \edef\bbl@tempb{\fontid\font}%
6057     \bblar@nofswarn
6058     \@nameuse{tracinglostchars}\z@
6059     \bblar@fetchjalt\bblar@elongated{}{from}{}%
6060     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
6061     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
6062     \addfontfeature{RawFeature+=jalt}%
6063     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
6064     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
6065     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
6066     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
6067     \directlua{%
6068       for k, v in pairs(Babel.arabic.from) do
6069         if Babel.arabic.dest[k] and
6070           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
6071           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
6072             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
6073         end
6074       end
6075     }%
6076   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

6077 \begingroup
6078 \catcode`#=11
6079 \catcode`~ =11
6080 \directlua{
6081
6082 Babel.arabic = Babel.arabic or {}
6083 Babel.arabic.from = {}
6084 Babel.arabic.dest = {}
6085 Babel.arabic.justify_factor = 0.95

```

```

6086 Babel.arabic.justify_enabled = true
6087 Babel.arabic.kashida_limit = -1
6088
6089 function Babel.arabic.justify(head)
6090   if not Babel.arabic.justify_enabled then return head end
6091   for line in node.traverse_id(node.id'hlist', head) do
6092     Babel.arabic.justify_hlist(head, line)
6093   end
6094   % In case the very first item is a line (eg, in \vbox):
6095   while head.prev do head = head.prev end
6096   return head
6097 end
6098
6099 function Babel.arabic.justify_hbox(head, gc, size, pack)
6100   local has_inf = false
6101   if Babel.arabic.justify_enabled and pack == 'exactly' then
6102     for n in node.traverse_id(12, head) do
6103       if n.stretch_order > 0 then has_inf = true end
6104     end
6105     if not has_inf then
6106       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
6107     end
6108   end
6109   return head
6110 end
6111
6112 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
6113   local d, new
6114   local k_list, k_item, pos_inline
6115   local width, width_new, full, k_curr, wt_pos, goal, shift
6116   local subst_done = false
6117   local elong_map = Babel.arabic.elong_map
6118   local cnt
6119   local last_line
6120   local GLYPH = node.id'glyph'
6121   local KASHIDA = Babel.attr_kashida
6122   local LOCALE = Babel.attr_locale
6123
6124   if line == nil then
6125     line = {}
6126     line.glue_sign = 1
6127     line.glue_order = 0
6128     line.head = head
6129     line.shift = 0
6130     line.width = size
6131   end
6132
6133   % Exclude last line.
6134   if ((line.next ~= nil or line.glue_sign == 1) and line.glue_order == 0) then
6135     elongs = {} % Stores elongated candidates of each line
6136     k_list = {} % And all letters with kashida
6137     pos_inline = 0 % Not yet used
6138
6139     for n in node.traverse_id(GLYPH, line.head) do
6140       pos_inline = pos_inline + 1 % To find where it is. Not used.
6141
6142       % Elongated glyphs
6143       if elong_map then
6144         local locale = node.get_attribute(n, LOCALE)
6145         if elong_map[locale] and elong_map[locale][n.font] and
6146           elong_map[locale][n.font][n.char] then
6147           table.insert(elongs, {node = n, locale = locale} )
6148           node.set_attribute(n.prev, KASHIDA, 0)

```

```

6149         end
6150     end
6151
6152     % Tatwil. First create a list of nodes marked with kashida. The
6153     % rest of nodes can be ignored. The list of used weights is build
6154     % when transforms with the key kashida= are declared.
6155     if Babel.kashida_wts then
6156         local k_wt = node.get_attribute(n, KASHIDA)
6157         if k_wt > 0 then % todo. parameter for multi inserts
6158             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
6159         end
6160     end
6161
6162 end % of node.traverse_id
6163
6164 if #elongs == 0 and #k_list == 0 then goto next_line end
6165 full = line.width
6166 shift = line.shift
6167 goal = full * Babel.arabic.justify_factor % A bit crude
6168 width = node.dimensions(line.head) % The 'natural' width
6169
6170 % == Elongated ==
6171 % Original idea taken from 'chickenize'
6172 while (#elongs > 0 and width < goal) do
6173     subst_done = true
6174     local x = #elongs
6175     local curr = elongs[x].node
6176     local oldchar = curr.char
6177     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6178     width = node.dimensions(line.head) % Check if the line is too wide
6179     % Substitute back if the line would be too wide and break:
6180     if width > goal then
6181         curr.char = oldchar
6182         break
6183     end
6184     % If continue, pop the just substituted node from the list:
6185     table.remove(elongs, x)
6186 end
6187
6188 % == Tatwil ==
6189 % Traverse the kashida node list so many times as required, until
6190 % the line is filled. The first pass adds a tatweel after each
6191 % node with kashida in the line, the second pass adds another one,
6192 % and so on. In each pass, add first the kashida with the highest
6193 % weight, then with lower weight and so on.
6194 if #k_list == 0 then goto next_line end
6195
6196 width = node.dimensions(line.head) % The 'natural' width
6197 k_curr = #k_list % Traverse backwards, from the end
6198 wt_pos = 1
6199
6200 while width < goal do
6201     subst_done = true
6202     k_item = k_list[k_curr].node
6203     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6204         d = node.copy(k_item)
6205         d.char = 0x0640
6206         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6207         d.xoffset = 0
6208         line.head, new = node.insert_after(line.head, k_item, d)
6209         width_new = node.dimensions(line.head)
6210         if width > goal or width == width_new then
6211             node.remove(line.head, new) % Better compute before

```

```

6212         break
6213     end
6214     if Babel.fix_diacr then
6215         Babel.fix_diacr(k_item.next)
6216     end
6217     width = width_new
6218 end
6219 if k_curr == 1 then
6220     k_curr = #k_list
6221     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6222 else
6223     k_curr = k_curr - 1
6224 end
6225 end
6226
6227 % Limit the number of tatweel by removing them. Not very efficient,
6228 % but it does the job in a quite predictable way.
6229 if Babel.arabic.kashida_limit > -1 then
6230     cnt = 0
6231     for n in node.traverse_id(GLYPH, line.head) do
6232         if n.char == 0x0640 then
6233             cnt = cnt + 1
6234             if cnt > Babel.arabic.kashida_limit then
6235                 node.remove(line.head, n)
6236             end
6237         else
6238             cnt = 0
6239         end
6240     end
6241 end
6242
6243 ::next_line::
6244
6245 % Must take into account marks and ins, see luatex manual.
6246 % Have to be executed only if there are changes. Investigate
6247 % what's going on exactly.
6248 if subst_done and not gc then
6249     d = node.hpack(line.head, full, 'exactly')
6250     d.shift = shift
6251     node.insert_before(head, line, d)
6252     node.remove(head, line)
6253 end
6254 end % if process line
6255 end
6256 }
6257 \endgroup
6258 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

10.9. Common stuff

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

```

6259 \def\bbl@scr@node@list{%
6260   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6261   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6262 \ifnum\bbl@bidimode=102 % bidi-r
6263   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6264 \fi
6265 \def\bbl@set@renderer{%
6266   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%

```

```

6267 \ifin@
6268 \let\bbl@unset@renderer\relax
6269 \else
6270 \bbl@exp{%
6271 \def\\bbl@unset@renderer{%
6272 \def\<g__fontspec_default_fontopts_clist>{%
6273 \[g__fontspec_default_fontopts_clist]}}%
6274 \def\<g__fontspec_default_fontopts_clist>{%
6275 Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
6276 \fi}
6277 <@Font selection@>

```

10.10 Automatic fonts and ids switching

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

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.

```

6278 \directlua{% DL6
6279 Babel.script_blocks = {
6280 ['dflt'] = {},
6281 ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6282             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFD}, {0x1EE00, 0x1EEFF}},
6283 ['Armn'] = {{0x0530, 0x058F}},
6284 ['Beng'] = {{0x0980, 0x09FF}},
6285 ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6286 ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6287 ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6288            {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6289 ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6290 ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6291            {0xAB00, 0xAB2F}},
6292 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6293 % Don't follow strictly Unicode, which places some Coptic letters in
6294 % the 'Greek and Coptic' block
6295 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6296 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6297            {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6298            {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6299            {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6300            {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6301            {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6302 ['Hebr'] = {{0x0590, 0x05FF},
6303            {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6304 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6305            {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6306 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6307 ['Knda'] = {{0x0C80, 0x0CFF}},
6308 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6309            {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6310            {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6311 ['Lao'] = {{0x0E80, 0x0EFF}},
6312 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6313            {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6314            {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6315 ['Mahj'] = {{0x11150, 0x1117F}},

```

```

6316 ['Mlym'] = {{0x0D00, 0x0D7F}},
6317 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6318 ['Orya'] = {{0x0B00, 0x0B7F}},
6319 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6320 ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6321 ['Taml'] = {{0x0B80, 0x0BFF}},
6322 ['Telu'] = {{0x0C00, 0x0C7F}},
6323 ['Tfng'] = {{0x2D30, 0x2D7F}},
6324 ['Thai'] = {{0x0E00, 0x0E7F}},
6325 ['Tibt'] = {{0x0F00, 0x0FFF}},
6326 ['Vaii'] = {{0xA500, 0xA63F}},
6327 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6328 }
6329
6330 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6331 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6332 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6333
6334 function Babel.locale_map(head)
6335   if not Babel.locale_mapped then return head end
6336
6337   local LOCALE = Babel.attr_locale
6338   local GLYPH = node.id('glyph')
6339   local inmath = false
6340   local toloc_save
6341   for item in node.traverse(head) do
6342     local toloc
6343     if not inmath and item.id == GLYPH then
6344       % Optimization: build a table with the chars found
6345       if Babel.chr_to_loc[item.char] then
6346         toloc = Babel.chr_to_loc[item.char]
6347       else
6348         for lc, maps in pairs(Babel.loc_to_scr) do
6349           for _, rg in pairs(maps) do
6350             if item.char >= rg[1] and item.char <= rg[2] then
6351               Babel.chr_to_loc[item.char] = lc
6352               toloc = lc
6353               break
6354             end
6355           end
6356         end
6357         % Treat composite chars in a different fashion, because they
6358         % 'inherit' the previous locale.
6359         if (item.char >= 0x0300 and item.char <= 0x036F) or
6360            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6361            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6362           Babel.chr_to_loc[item.char] = -2000
6363           toloc = -2000
6364         end
6365         if not toloc then
6366           Babel.chr_to_loc[item.char] = -1000
6367         end
6368       end
6369       if toloc == -2000 then
6370         toloc = toloc_save
6371       elseif toloc == -1000 then
6372         toloc = nil
6373       end
6374       if toloc and Babel.locale_props[toloc] and
6375          Babel.locale_props[toloc].letters and
6376          tex.getcatcode(item.char) \string~= 11 then
6377         toloc = nil
6378       end

```

```

6379     if toloc and Babel.locale_props[toloc].script
6380         and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6381         and Babel.locale_props[toloc].script ==
6382             Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6383         toloc = nil
6384     end
6385     if toloc then
6386         if Babel.locale_props[toloc].lg then
6387             item.lang = Babel.locale_props[toloc].lg
6388             node.set_attribute(item, LOCALE, toloc)
6389         end
6390         if Babel.locale_props[toloc]['/'..item.font] then
6391             item.font = Babel.locale_props[toloc]['/'..item.font]
6392         end
6393     end
6394     toloc_save = toloc
6395     elseif not inmath and item.id == 7 then % Apply recursively
6396         item.replace = item.replace and Babel.locale_map(item.replace)
6397         item.pre      = item.pre and Babel.locale_map(item.pre)
6398         item.post     = item.post and Babel.locale_map(item.post)
6399     elseif item.id == node.id'math' then
6400         inmath = (item.subtype == 0)
6401     end
6402 end
6403 return head
6404 end
6405 }

```

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

```

6406 \newcommand\babelcharproperty[1]{%
6407   \count@=#1\relax
6408   \ifvmode
6409     \expandafter\bbl@chprop
6410   \else
6411     \bbl@error{charproperty-only-vertical}{}{}{}%
6412   \fi}
6413 \newcommand\bbl@chprop[3][\the\count@]{%
6414   \@tempcnta=#1\relax
6415   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6416   {\bbl@error{unknown-char-property}{}{#2}{}%
6417   }%
6418   \loop
6419     \bbl@cs{chprop@#2}{#3}%
6420   \ifnum\count@< \@tempcnta
6421     \advance\count@\@ne
6422   \repeat}
6423 %
6424 \def\bbl@chprop@direction#1{%
6425   \directlua{
6426     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6427     Babel.characters[\the\count@]['d'] = '#1'
6428   }}
6429 \let\bbl@chprop@bc\bbl@chprop@direction
6430 %
6431 \def\bbl@chprop@mirror#1{%
6432   \directlua{
6433     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6434     Babel.characters[\the\count@]['m'] = '\number#1'
6435   }}
6436 \let\bbl@chprop@bmg\bbl@chprop@mirror
6437 %
6438 \def\bbl@chprop@linebreak#1{%

```

```

6439 \directlua{
6440   Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6441   Babel.cjk_characters[\the\count@]['c'] = '#1'
6442 }
6443 \let\bbl@chprop@lb\bbl@chprop@linebreak
6444 %
6445 \def\bbl@chprop@locale#1{%
6446   \directlua{
6447     Babel.chr_to_loc = Babel.chr_to_loc or {}
6448     Babel.chr_to_loc[\the\count@] =
6449       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6450   }

```

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.

```

6451 \directlua{% DL7
6452   Babel.nohyphenation = \the\l@nohyphenation
6453 }

```

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

```

6454 \begingroup
6455 \catcode`\~ = 12
6456 \catcode`\% = 12
6457 \catcode`\& = 14
6458 \catcode`\| = 12
6459 \gdef\babelprehyphenation{%&%
6460   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6461 \gdef\babelposthyphenation{%&%
6462   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6463 %
6464 \gdef\bbl@settransform#1[#2]#3#4#5{%&%
6465   \ifcase#1
6466     \bbl@activateprehyphen
6467   \or
6468     \bbl@activateposthyphen
6469   \fi
6470 \begingroup
6471   \def\babeltempa{\bbl@add@list\babeltempb}%&%
6472   \let\babeltempb\empty
6473   \def\bbl@tempa{#5}%&%
6474   \bbl@replace\bbl@tempa{,}{ ,}%&% TODO. Ugly trick to preserve {}
6475   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&%
6476     \bbl@ifsamestring{##1}{remove}%&%
6477     {\bbl@add@list\babeltempb{nil}}}%&%
6478     {\directlua{
6479       local rep = [= [#1] =]
6480       local three_args = '%s*=%s*([%- %d%.%a{ }|]+)%s+([%- %d%.%a{ }|]+)%s+([%- %d%.%a{ }|]+)'
6481       &% Numeric passes directly: kern, penalty...
6482       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6483       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6484       rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6485       rep = rep:gsub('(string)%s*=%s*([%-s,]*)', Babel.capture_func)
6486       rep = rep:gsub('node%s*=%s*([%-+)%s*([%-a]*)', Babel.capture_node)
6487       rep = rep:gsub('(norule)' .. three_args,
6488         'norule = {' .. '%2, %3, %4' .. '}')
6489       if #1 == 0 or #1 == 2 then

```

```

6490     rep = rep:gsub( '(space)' .. three_args,
6491       'space = {' .. '%2, %3, %4' .. '}' )
6492     rep = rep:gsub( '(spacefactor)' .. three_args,
6493       'spacefactor = {' .. '%2, %3, %4' .. '}' )
6494     rep = rep:gsub( '(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6495     &% Transform values
6496     rep, n = rep:gsub( '{([%a%-%.]+)|([%a%_%.]+)}',
6497       function(v,d)
6498         return string.format (
6499           '{\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6500           v,
6501           load( 'return Babel.locale_props'..
6502             '\the\csname bbl@id@@#3\endcsname].' .. d)() )
6503         end )
6504     rep, n = rep:gsub( '{([%a%-%.]+)|([%-d%.]+)}',
6505       '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6506   end
6507   if #1 == 1 then
6508     rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6509     rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6510     rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6511   end
6512   tex.print([[string\babeltempa{[] .. rep .. [{}]])
6513   ]]}&%
6514 \bbl@foreach\babeltempb{&%
6515   \bbl@forkv{##1}}{&%
6516     \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6517       post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6518     \ifin\else
6519       \bbl@error{bad-transform-option}{###1}{}&%
6520     \fi}&%
6521 \let\bbl@kv@attribute\relax
6522 \let\bbl@kv@label\relax
6523 \let\bbl@kv@fonts\empty
6524 \let\bbl@kv@prepend\relax
6525 \bbl@forkv{#2}{\bbl@csarg\edef{kv{##1}{##2}}&%
6526 \ifx\bbl@kv@fonts\empty\else\bbl@settransfont\fi
6527 \ifx\bbl@kv@attribute\relax
6528   \ifx\bbl@kv@label\relax\else
6529     \bbl@exp{\\bbl@trim@def\\bbl@kv@fonts{\bbl@kv@fonts}}&%
6530     \bbl@replace\bbl@kv@fonts{ }{,}&%
6531     \edef\bbl@kv@attribute{\bbl@ATR@{\bbl@kv@label @#3@{\bbl@kv@fonts}}&%
6532     \count@ \z@
6533     \def\bbl@elt##1##2##3{&%
6534       \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6535       {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6536         {\count@ \@ne}&%
6537         {\bbl@error{font-conflict-transforms}{}}{}}}&%
6538       }&%
6539     \bbl@transfont@list
6540     \ifnum\count@=\z@
6541       \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6542         {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6543     \fi
6544     \bbl@ifunset{\bbl@kv@attribute}&%
6545     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6546     {}&%
6547     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6548   \fi
6549 \else
6550   \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6551 \fi
6552 \directlua{

```

```

6553     local lbkr = Babel.linebreaking.replacements[#1]
6554     local u = unicode.utf8
6555     local id, attr, label
6556     if #1 == 0 then
6557         id = \the\csname bbl@id@@#3\endcsname\space
6558     else
6559         id = \the\csname l@#3\endcsname\space
6560     end
6561     \ifx\bbl@kv@attribute\relax
6562         attr = -1
6563     \else
6564         attr = luatexbase.registernumber'\bbl@kv@attribute'
6565     \fi
6566     \ifx\bbl@kv@label\relax\else &% Same refs:
6567         label = [==[\bbl@kv@label]==]
6568     \fi
6569     &% Convert pattern:
6570     local patt = string.gsub([==[#4]==], '%s', '')
6571     if #1 == 0 then
6572         patt = string.gsub(patt, '|', ' ')
6573     end
6574     if not u.find(patt, '()', nil, true) then
6575         patt = '()' .. patt .. '()'
6576     end
6577     patt = string.gsub(patt, '%(%)%^', '^()')
6578     patt = string.gsub(patt, '%$(%)', '()$')
6579     patt = u.gsub(patt, '{(.)}',
6580         function (n)
6581             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6582         end)
6583     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6584         function (n)
6585             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6586         end)
6587     lbkr[id] = lbkr[id] or {}
6588     table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6589         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6590     }&%
6591 \endgroup}
6592 \endgroup
6593 %
6594 \let\bbl@transfont@list\@empty
6595 \def\bbl@settransfont{%
6596     \global\let\bbl@settransfont\relax % Execute only once
6597     \gdef\bbl@transfont{%
6598         \def\bbl@elt####1####2####3{%
6599             \bbl@ifblank{####3}%
6600                 {\count@tw@}% Do nothing if no fonts
6601                 {\count@z@
6602                     \bbl@vforeach{####3}{%
6603                         \def\bbl@tempd{#####1}%
6604                         \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6605                         \ifx\bbl@tempd\bbl@tempe
6606                             \count@\@ne
6607                         \else\ifx\bbl@tempd\bbl@transfam
6608                             \count@\@ne
6609                         \fi\fi}%
6610                     \ifcase\count@
6611                         \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6612                     \or
6613                         \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6614                     \fi}%
6615                 \bbl@transfont@list}%

```

```

6616 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6617 \gdef\bbl@transfam{-unknown-}%
6618 \bbl@foreach\bbl@font@fams{%
6619   \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6620   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6621   {\xdef\bbl@transfam{##1}}%
6622   {}}
6623 %
6624 \DeclareRobustCommand\enablelocaletransform[1]{%
6625   \bbl@ifunset\bbl@ATR@#1@\language @}%
6626   {\bbl@error{transform-not-available}{#1}{}}%
6627   {\bbl@csarg\setattribute{ATR@#1@\language @}{\@ne}}
6628 \DeclareRobustCommand\disablelocaletransform[1]{%
6629   \bbl@ifunset\bbl@ATR@#1@\language @}%
6630   {\bbl@error{transform-not-available-b}{#1}{}}%
6631   {\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`.

```

6632 \def\bbl@activateposthyphen{%
6633   \let\bbl@activateposthyphen\relax
6634   \ifx\bbl@attr@hboxed\undefined
6635     \newattribute\bbl@attr@hboxed
6636   \fi
6637   \directlua{
6638     require('babel-transforms.lua')
6639     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6640   }}
6641 \def\bbl@activateprehyphen{%
6642   \let\bbl@activateprehyphen\relax
6643   \ifx\bbl@attr@hboxed\undefined
6644     \newattribute\bbl@attr@hboxed
6645   \fi
6646   \directlua{
6647     require('babel-transforms.lua')
6648     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6649   }}
6650 \newcommand\SetTransformValue[3]{%
6651   \directlua{
6652     Babel.locale_props[\the\csname bbl@id@@#1\endcsname].vars["#2"] = #3
6653   }}

```

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.

```

6654 \newcommand\ShowBabelTransforms[1]{%
6655   \bbl@activateprehyphen
6656   \bbl@activateposthyphen
6657   \begingroup
6658     \directlua{ Babel.show_transforms = true }%
6659     \setbox\z@\vbox{#1}%
6660     \directlua{ Babel.show_transforms = false }%
6661   \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.

```

6662 \newcommand\localeprehyphenation[1]{%
6663   \directlua{ Babel.string_prehyphenation([==#1==], \the\localeid) }}

```

10.11.Bidi

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

```
6664 \def\bbl@activate@preotf{%
6665   \let\bbl@activate@preotf\relax % only once
6666   \directlua{
6667     function Babel.pre_otfload_v(head)
6668       if Babel.numbers and Babel.digits_mapped then
6669         head = Babel.numbers(head)
6670       end
6671       if Babel.bidi_enabled then
6672         head = Babel.bidi(head, false, dir)
6673       end
6674       return head
6675     end
6676     %
6677     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6678       if Babel.numbers and Babel.digits_mapped then
6679         head = Babel.numbers(head)
6680       end
6681       if Babel.bidi_enabled then
6682         head = Babel.bidi(head, false, dir)
6683       end
6684       return head
6685     end
6686     %
6687     luatexbase.add_to_callback('pre_linebreak_filter',
6688       Babel.pre_otfload_v,
6689       'Babel.pre_otfload_v',
6690       Babel.priority_in_callback('pre_linebreak_filter',
6691         'luaotfload.node_processor') or nil)
6692     %
6693     luatexbase.add_to_callback('hpack_filter',
6694       Babel.pre_otfload_h,
6695       'Babel.pre_otfload_h',
6696       Babel.priority_in_callback('hpack_filter',
6697         'luaotfload.node_processor') or nil)
6698   }}
```

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`.

```
6699 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6700   \let\bbl@beforeforeign\leavevmode
6701   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6702   \RequirePackage{luatexbase}
6703   \bbl@activate@preotf
6704   \directlua{
6705     require('babel-data-bidi.lua')
6706     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6707       require('babel-bidi-basic.lua')
6708     \or
6709       require('babel-bidi-basic-r.lua')
6710     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6711     table.insert(Babel.ranges, {0xF0000, 0xFFFFFD, 'on'})
6712     table.insert(Babel.ranges, {0x100000, 0x10FFFFD, 'on'})
6713   \fi}
6714   \newattribute\bbl@attr@dir
6715   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6716   \bbl@exp{\output{\bodydir\pagedir\the\output}}
```

```

6717 \fi
6718 %
6719 \chardef\bbl@thetextdir\z@
6720 \chardef\bbl@thepardir\z@
6721 \def\bbl@setluadir#1#2{% 1=\text/pardirection 2= 0:l 1:r 2:a}
6722 \ifcase#2\relax
6723 \ifcase#1\else#1=\z@\fi
6724 \else
6725 \ifcase#1#1=\@ne\fi
6726 \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.

6727 \def\bbl@thedir{0}
6728 \def\bbl@textdir#1{%
6729 \bbl@setluadir\textdirection{#1}%
6730 \chardef\bbl@thetextdir#1\relax
6731 \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6732 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6733 \def\bbl@pardir#1{% Used twice
6734 \bbl@setluadir\pardirection{#1}%
6735 \chardef\bbl@thepardir#1\relax}
6736 \def\bbl@bodydir{\bbl@setluadir\bodydirection}% Used once
6737 \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.

6738 \ifnum\bbl@bidimode>\z@ % Any bidi=
6739 \def\bbl@insidemath{0}%
6740 \def\bbl@everymath{\def\bbl@insidemath{1}}
6741 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6742 \frozen@everymath\expandafter{%
6743 \expandafter\bbl@everymath\the\frozen@everymath}
6744 \frozen@everydisplay\expandafter{%
6745 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6746 \AtBeginDocument{
6747 \directlua{
6748 function Babel.math_box_dir(head)
6749 if not (token.get_macro('bbl@insidemath') == '0') then
6750 if Babel.hlist_has_bidi(head) then
6751 local d = node.new(node.id'dir')
6752 d.dir = '+TRT'
6753 for item in node.traverse(head) do
6754 if item.id == 11 or item.id == node.id'glyph' then
6755 head = node.insert_before(head, item, d)
6756 break
6757 end
6758 end
6759 local inmath = false
6760 for item in node.traverse(head) do
6761 if item.id == 11 then
6762 inmath = (item.subtype == 0)
6763 elseif not inmath then
6764 node.set_attribute(item,
6765 Babel.attr_dir, token.get_macro('bbl@thedir'))
6766 end
6767 end
6768 end
6769 end
6770 return head
6771 end
6772 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6773 "Babel.math_box_dir", 0)
6774 if Babel.unset_atdir then

```

```

6775     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6776     "Babel.unset_atdir")
6777     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6778     "Babel.unset_atdir")
6779     end
6780     luatexbase.add_to_callback("post_mlist_to_hlist_filter", function(head)
6781     local dir = tex.mathdirection
6782     local n = node.new("dir")
6783     n.direction = dir
6784     head = node.insert_before(head, head, n)
6785     n = node.new("dir", 1)
6786     n.direction = dir
6787     head = node.insert_after(head, node.tail(head), n)
6788     return head
6789     end, "Babel.ensure_math_dir")
6790 } }%
6791 \fi

Experimental. Tentative name.

6792 \DeclareRobustCommand\localebox[1]{%
6793 {\def\bbl@insidemath{0}%
6794 \mbox{\foreignlanguage{\language}\language{#1}}}}

```

10.12 Layout

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

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

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

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

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

```

6795 \bbl@trace{Redefinitions for bidi layout}
6796 %
6797 <<(*More package options)>> ≡
6798 \chardef\bbl@eqnpos\z@
6799 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6800 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6801 <</More package options>>

```

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

```

6802 \ifnum\bbl@bidimode>\z@ % Any bidi=
6803 \matheqdirmode\@ne
6804 \mathemptydisplaymode\@ne
6805 \breakafterdirmode\@ne
6806 \fixupboxesmode\@ne
6807 \let\bbl@eqnodir\relax
6808 \def\bbl@eqdel{()}
6809 \def\bbl@eqnum{%
6810 {\normalfont\normalcolor
6811 \expandafter\@firstoftwo\bbl@eqdel

```

```

6812 \theequation
6813 \expandafter\@secondoftwo\bbledqdel}}
6814 \def\bbledputeqno#1{\eqno\hbox{#1}}
6815 \def\bbledputleqno#1{\leqno\hbox{#1}}
6816 \def\bbledeqno@flip#1{% So
6817 \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6818 \eqno
6819 \hb@xt@.01pt{%
6820 \hb@xt@\displaywidth{\hss#1\glet\bbledupset\@currentlabel}}\hss}%
6821 \else
6822 \leqno\hbox{#1\glet\bbledupset\@currentlabel}%
6823 \fi
6824 \bbledexp{\def\\@currentlabel{\bbledupset}}}}
6825 \def\bbledleqno@flip#1{%
6826 \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6827 \leqno
6828 \hb@xt@.01pt{%
6829 \hss\hb@xt@\displaywidth{\hss#1\glet\bbledupset\@currentlabel}\hss}}%
6830 \else
6831 \eqno\hbox{#1\glet\bbledupset\@currentlabel}%
6832 \fi
6833 \bbledexp{\def\\@currentlabel{\bbledupset}}}}
6834 %
6835 \AtBeginDocument{%
6836 \ifx\bblednoamsmath\relax\else
6837 \ifx\maketag@@@undefined % Normal equation, eqnarray
6838 \ifnum\bbledeqnpos=\tw@
6839 \bbledreplace\equation{\hb@xt@\linewidth}%
6840 {\hbox bdir\mathdirection to\linewidth}%
6841 \bbledcarg\bbledsreplace{[ ]}{\hb@xt@\linewidth}%
6842 {\hbox bdir\mathdirection to\linewidth}%
6843 \fi
6844 \AddToHook{env/equation/begin}{%
6845 \ifnum\bbledthetextdir>z@
6846 \let\@eqnnum\bbledeqnum
6847 \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6848 \chardef\bbledthetextdir\z@
6849 \bbledadd\normalfont{\bbledeqnodir}%
6850 \ifcase\bbledeqnpos
6851 \let\bbledputeqno\bbledeqno@flip
6852 \or
6853 \let\bbledputeqno\bbledleqno@flip
6854 \fi
6855 \fi}%
6856 \ifnum\bbledeqnpos=\tw@\else
6857 \def\endequation{\bbledputeqno{\@eqnnum}$$\@ignoretrue}%
6858 \fi
6859 \AddToHook{env/eqnarray/begin}{%
6860 \ifnum\bbledthetextdir>z@
6861 \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6862 \chardef\bbledthetextdir\z@
6863 \bbledadd\normalfont{\bbledeqnodir}%
6864 \ifnum\bbledeqnpos=\@ne
6865 \def\@eqnnum{%
6866 \setbox\z@\hbox{\bbledeqnum}%
6867 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6868 \else
6869 \let\@eqnnum\bbledeqnum
6870 \fi
6871 \fi}
6872 \else % amstex
6873 \ifnum\bbledeqnpos=\tw@
6874 \bbledexp{% Hack to hide maybe undefined conditionals:

```

```

6875      \\bbl@sreplace\\multline@crcr{\<if@flegn>\tabskip\z@skip\<fi>\crcr}}%
6876 \fi
6877 \bbl@exp{% Hack to hide maybe undefined conditionals:
6878   \chardef\bbl@eqnpos=0%
6879   \<iftagsleft@>1\<else>\<if@flegn>2\<fi>\<fi>\relax}%
6880 \ifnum\bbl@eqnpos=\@ne
6881   \let\bbl@ams@lap\hbox
6882 \else
6883   \let\bbl@ams@lap\llap
6884 \fi
6885 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6886 \bbl@sreplace\intertext@\{normalbaselines}%
6887   {\normalbaselines
6888   \ifx\bbl@eqnodir\relax\else\bbl@paddir\@ne\bbl@eqnodir\fi}%
6889 \ExplSyntaxOff
6890 \def\bbl@ams@tagbox#1#2{\bbl@eqnodir#2}% #1=hbox|@lap|flip
6891 \ifx\bbl@ams@lap\hbox % leqno
6892   \def\bbl@ams@flip#1{%
6893     \hbox to 0.01pt{\hss\hbox to\displaywidth{\#1\hss}}}%
6894 \else % eqno
6895   \def\bbl@ams@flip#1{%
6896     \hbox to 0.01pt{\hbox to\displaywidth{\hss\#1}\hss}}%
6897 \fi
6898 \def\bbl@ams@preset#1{%
6899   \ifnum\bbl@thetextdir>\z@
6900     \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6901     \bbl@sreplace\textdef@\{\hbox}\{\bbl@ams@tagbox\hbox}%
6902     \bbl@sreplace\maketag@@@\{\hbox}\{\bbl@ams@tagbox#1}%
6903   \fi}%
6904 \def\bbl@ams@equation{%
6905   \ifnum\bbl@thetextdir>\z@
6906     \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6907     \chardef\bbl@thetextdir\z@
6908     \bbl@add\normalfont{\bbl@eqnodir}%
6909     \ifcase\bbl@eqnpos
6910       \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6911     \or
6912       \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6913     \fi
6914   \fi}%
6915 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6916 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6917 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6918 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6919 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6920 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6921 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6922 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6923 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6924 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6925 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6926 % Hackish, for proper alignment. Don't ask me why it works!:
6927 \bbl@exp{% Avoid a 'visible' conditional
6928   \\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6929   \\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6930 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6931 \AddToHook{env/split/before}{%
6932   \ifnum\bbl@thetextdir>\z@
6933     \bbl@ifsamestring\@currentenv{equation}%
6934     {\ifx\bbl@ams@lap\hbox % leqno
6935       \def\bbl@ams@flip#1{%
6936         \hbox to 0.01pt{\hbox to\displaywidth{\#1\hss}\hss}}%
6937     \else

```

```

6938         \def\bbl@ams@flip#1{%
6939             \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6940         \fi}%
6941     }%
6942 \fi}%
6943 \fi\fi}
6944 \fi

Declarations specific to lua, called by \babelprovide.

6945 \def\bbl@provide@extra#1{%
6946     % == onchar ==
6947     \ifx\bbl@KVP@onchar\@nnil\else
6948         \bbl@luahyphenate
6949         \bbl@exp{%
6950             \\AddToHook{env/document/before}{%
6951                 {\let\\bbl@ifrestoring\\@firstoftwo
6952                     \\select@language{#1}{}}}%
6953         \directlua{
6954             if Babel.locale_mapped == nil then
6955                 Babel.locale_mapped = true
6956                 Babel.linebreaking.add_before(Babel.locale_map, 1)
6957                 Babel.loc_to_scr = {}
6958                 Babel.chr_to_loc = Babel.chr_to_loc or {}
6959             end
6960             Babel.locale_props[\the\localeid].letters = false
6961         }%
6962         \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6963         \ifin@
6964             \directlua{
6965                 Babel.locale_props[\the\localeid].letters = true
6966             }%
6967         \fi
6968         \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6969         \ifin@
6970             \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6971                 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6972             \fi
6973             \bbl@exp{\\bbl@add\\bbl@starthyphens
6974                 {\bbl@patterns@lua{\languagename}}}%
6975             \directlua{
6976                 if Babel.script_blocks['\bbl@cl{sbc}'] then
6977                     Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6978                     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
6979                 end
6980             }%
6981         \fi
6982         \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6983         \ifin@
6984             \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{%
6985             \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{%
6986             \directlua{
6987                 if Babel.script_blocks['\bbl@cl{sbc}'] then
6988                     Babel.loc_to_scr[\the\localeid] =
6989                     Babel.script_blocks['\bbl@cl{sbc}']
6990                 end}%
6991             \ifx\bbl@mapselect\@undefined
6992                 \AtBeginDocument{%
6993                     \bbl@patchfont{\bbl@mapselect}}%
6994                     {\selectfont}}%
6995             \def\bbl@mapselect{%
6996                 \let\bbl@mapselect\relax
6997                 \edef\bbl@prefontid{\fontid\font}}%
6998             \def\bbl@mapdir##1{%

```

```

6999     \begingroup
7000     \setbox\z@\hbox{% Force text mode
7001     \def\language{##1}%
7002     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
7003     \bbl@switchfont
7004     \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
7005     \directlua{
7006         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
7007         ['/\bbl@prefontid'] = \fontid\font\space}%
7008     \fi}%
7009     \endgroup}%
7010     \fi
7011     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7012     \fi
7013     \fi
7014     % == mapfont ==
7015     % For bidi texts, to switch the font based on direction. Deprecated
7016     \ifx\bbl@KVP\mapfont\@nnil\else
7017     \bbl@ifsamestring{\bbl@KVP\mapfont}{direction}}{%
7018     {\bbl@error{unknown-mapfont}}}%
7019     \bbl@ifunset{\bbl@lsys\language}{\bbl@provide\lsys{\language}}{%
7020     \bbl@ifunset{\bbl@wdir\language}{\bbl@provide\dirs{\language}}{%
7021     \ifx\bbl@mapselect\undefined
7022     \AtBeginDocument{%
7023     \bbl@patchfont{\bbl@mapselect}}%
7024     {\selectfont}}%
7025     \def\bbl@mapselect{%
7026     \let\bbl@mapselect\relax
7027     \edef\bbl@prefontid{\fontid\font}}%
7028     \def\bbl@mapdir##1{%
7029     {\def\language{##1}%
7030     \let\bbl@ifrestoring\@firstoftwo % avoid font warning
7031     \bbl@switchfont
7032     \directlua{Babel.fontmap
7033     [\the\csname bbl@wdir@##1\endcsname]%
7034     [\bbl@prefontid]=\fontid\font}}}%
7035     \fi
7036     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7037     \fi
7038     % == Line breaking: CJK quotes ==
7039     \ifcase\bbl@engine\or
7040     \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}%
7041     \ifin@
7042     \bbl@ifunset{\bbl@quote\language}}{%
7043     {\directlua{
7044     Babel.locale_props[\the\localeid].cjk_quotes = {}
7045     local cs = 'op'
7046     for c in string.utfvalues(
7047     [[\csname bbl@quote\language\endcsname]]) do
7048     if Babel.cjk_characters[c].c == 'qu' then
7049     Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
7050     end
7051     cs = ( cs == 'op') and 'cl' or 'op'
7052     end
7053     }}%
7054     \fi
7055     \fi
7056     % == Counters: mapdigits ==
7057     % Native digits
7058     \ifx\bbl@KVP\mapdigits\@nnil\else
7059     \bbl@ifunset{\bbl@dgnat\language}}{%
7060     {\bbl@activate@preotf
7061     \directlua{

```

```

7062     Babel.digits_mapped = true
7063     Babel.digits = Babel.digits or {}
7064     Babel.digits[\the\localeid] =
7065         table.pack(string.utfvalue('\bbl@cl{dgnat}'))
7066     if not Babel.numbers then
7067         function Babel.numbers(head)
7068             local LOCALE = Babel.attr_locale
7069             local GLYPH = node.id'glyph'
7070             local inmath = false
7071             for item in node.traverse(head) do
7072                 if not inmath and item.id == GLYPH then
7073                     local temp = node.get_attribute(item, LOCALE)
7074                     if Babel.digits[temp] then
7075                         local chr = item.char
7076                         if chr > 47 and chr < 58 then
7077                             item.char = Babel.digits[temp][chr-47]
7078                         end
7079                     end
7080                     elseif item.id == node.id'math' then
7081                         inmath = (item.subtype == 0)
7082                     end
7083                 end
7084             return head
7085         end
7086     end
7087 }}%
7088 \fi
7089 % == transforms ==
7090 \ifx\bbl@KVP@transforms\@nnil\else
7091 \def\bbl@elt##1##2##3{%
7092     \in@{$transforms.}{$##1}%
7093     \ifin@
7094     \def\bbl@tempa{##1}%
7095     \bbl@replace\bbl@tempa{transforms.}{}%
7096     \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
7097     \fi}%
7098 \bbl@exp{%
7099     \\bbl@ifblank{\bbl@cl{dgnat}}%
7100     {\let\\bbl@tempa\relax}%
7101     {\def\\bbl@tempa{%
7102         \\bbl@elt{transforms.prehyphenation}%
7103         {digits.native.1.0}{([0-9])}%
7104         \\bbl@elt{transforms.prehyphenation}%
7105         {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}%
7106 \ifx\bbl@tempa\relax\else
7107     \toks@\expandafter\expandafter\expandafter{%
7108         \csname bbl@inidata@\language\endcsname}%
7109     \bbl@carg\edef{inidata@\language}%
7110     \unexpanded\expandafter{\bbl@tempa}%
7111     \the\toks@}%
7112 \fi
7113 \csname bbl@inidata@\language\endcsname
7114 \bbl@release@transforms\relax % \relax closes the last item.
7115 \fi}

Start tabular here:

7116 \def\localerestoredirs{%
7117     \ifcase\bbl@thetextdir
7118     \ifnum\textdirection=\z@\else\textdirection=\z@\fi
7119     \else
7120     \ifnum\textdirection=\@ne\else\textdirection=\@ne\fi
7121     \fi
7122     \ifcase\bbl@thepardir

```

```

7123 \ifnum\pardirection=\z@else\pardirection=\z@\bodydirection=\z@\fi
7124 \else
7125 \ifnum\pardirection=\@neelse\pardirection=\@ne\bodydirection=\@ne\fi
7126 \fi}
7127 %
7128 \IfBabelLayout{tabular}%
7129 {\chardef\bbl@tabular@mode\z@}% All RTL
7130 {\IfBabelLayout{lrtabular}%
7131 {\chardef\bbl@tabular@mode\@ne}%
7132 {\chardef\bbl@tabular@mode\z@}}% Mixed, with LTR cols
7133 %
7134 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
7135 % Redefine: vrules mess up dirs (why?).
7136 \AtBeginDocument{\def\@arstrut{\relax\copy\@arstrutbox}}%
7137 \ifcase\bbl@tabular@modeelse % 1 = Mixed
7138 \let\bbl@parabefore\relax
7139 \AddToHook{para/before}{\bbl@parabefore}
7140 \AtBeginDocument{%
7141 \bbl@replace\@tabular{\hbox\bgroup}{\hbox bdir\z@\bgroup}%
7142 \ifnum\bbl@tabular@mode=\@ne
7143 \bbl@ifunset{\@tabclassz}{\}%
7144 \bbl@exp{% Hide conditionals
7145 \\\bbl@sreplace\\\@tabclassz
7146 {\<ifcase>\\\@chnum}%
7147 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
7148 \@ifpackageloaded{colortbl}%
7149 {\bbl@sreplace\@classz
7150 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
7151 {\@ifpackageloaded{array}%
7152 {\bbl@exp{% Hide conditionals
7153 \\\bbl@sreplace\\\@classz
7154 {\<ifcase>\\\@chnum}%
7155 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
7156 \\\bbl@sreplace\\\@classz
7157 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
7158 {}}}%
7159 \fi}%
7160 \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.

```

7161 \AtBeginDocument{%
7162 \@ifpackageloaded{multicol}%
7163 {\toks\expandafter{\multi@column@out}%
7164 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
7165 {}}%
7166 \@ifpackageloaded{paracol}%
7167 {\edef\pcol@output{%
7168 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7169 {}}%
7170 \fi

```

Finish here if there is no layout.

```
7171 \ifx\bbl@opt@layout\@nnil\endinput\fi
```

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

```

7172 \chardef\bbl@trace@vboxdir\z@
7173 \ifnum\bbl@bidimode>\z@ % Any bidi=
7174 \IfBabelLayout{nopars}{}
7175 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7176 \IfBabelLayout{pars}
7177 {\chardef\bbl@trace@vboxdir\@ne
7178 \def\@hangfrom#1{%

```

```

7179 \setbox\@tempboxa\hbox{\{#1\}}%
7180 \hangindent\wd\@tempboxa
7181 \ifnum\bb@vbox@bodydir=\pardirection\else
7182 \shapemode\@ne
7183 \fi
7184 \noindent\box\@tempboxa}}
7185 {}
7186 \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).

```

7187 \IfBabelLayout{lists}
7188 {\chardef\bb@trace@vboxdir\@ne
7189 \let\bb@OL@list\list
7190 \bb@Sreplace\list{\parshape}{\bb@listparshape}%
7191 \let\bb@NL@list\list
7192 \def\bb@listparshape#1#2#3{%
7193 \parshape #1 #2 #3 %
7194 \ifnum\bb@vbox@bodydir=\pardirection\else
7195 \shapemode\tw@
7196 \fi}}
7197 {}
7198 %
7199 \ifcase\bb@trace@vboxdir\else
7200 \AtBeginDocument{\chardef\bb@vbox@bodydir\pagedirection}%
7201 \def\bb@vbox@lists{\chardef\bb@vbox@bodydir\bodydirection}%
7202 \let\bb@bidi@vbox\everyvbox
7203 \@nameuse{newtoks}\everyvbox % \outer in Plain
7204 \everyvbox\expandafter{\the\bb@bidi@vbox}%
7205 \bb@bidi@vbox{\bb@vbox@lists\the\everyvbox}%
7206 \fi
7207 %
7208 \IfBabelLayout{graphics}
7209 {\let\bb@pictresetdir\relax
7210 \def\bb@pictsetdir#1{%
7211 \ifcase\bb@thetextdir
7212 \let\bb@pictresetdir\relax
7213 \else
7214 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7215 \or\textdir TLT
7216 \else\bodydir TLT \textdir TLT
7217 \fi
7218 % \text\par\dir required in pgf:
7219 \def\bb@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7220 \fi}%
7221 \AddToHook{env/picture/begin}{\bb@pictsetdir\tw@}%
7222 \directlua{
7223 Babel.get_picture_dir = true
7224 Babel.picture_has_bidi = 0
7225 %
7226 function Babel.picture_dir (head)
7227 if not Babel.get_picture_dir then return head end
7228 if Babel.hlist_has_bidi(head) then
7229 Babel.picture_has_bidi = 1
7230 end
7231 return head
7232 end
7233 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7234 "Babel.picture_dir")
7235 }%
7236 \AddToHook{package/graphics/after}{%
7237 \bb@exp{%
7238 \\bb@Sreplace\\rotatebox{\hbox{\string#2}}

```

```

7239         {\hbox bdir\z@\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>{\string#2}}}}
7240 \AddToHook{package/graphicsx/after}{%
7241   \bbl@exp{%
7242     \\bbl@sreplace\\Grot@box@std{\hbox{\string#2}}
7243     {\hbox bdir\z@\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>{\string#2}}}%
7244     \\bbl@sreplace\\Grot@box@kv{\hbox{\string#3}}
7245     {\hbox bdir\z@\hbox bdir\<ifcase>\bbl@thetextdir\z@\<else>\@ne\<fi>{\string#3}}}%
7246     \\bbl@sreplace\\Grot@box{\hbox}{\hbox bdir\z@}}
7247 \AtBeginDocument{%
7248   \long\def\put(#1,#2)#3{%
7249     \@killglue
7250     % Try:
7251     \ifx\bbl@pictresetdir\relax
7252       \def\bbl@tempc{0}%
7253     \else
7254       \directlua{
7255         Babel.get_picture_dir = true
7256         Babel.picture_has_bidi = 0
7257       }%
7258       \setbox\z@\hb@xt\z@{%
7259         \@defaultunitsset\@tempdimc{#1}\unitlength
7260         \kern\@tempdimc
7261         #3\hss}%
7262       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7263     \fi
7264     % Do:
7265     \@defaultunitsset\@tempdimc{#2}\unitlength
7266     \raise\@tempdimc\hb@xt\z@{%
7267       \@defaultunitsset\@tempdimc{#1}\unitlength
7268       \kern\@tempdimc
7269       {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
7270     \ignorespaces}%
7271   \MakeRobust\put}%
7272 \AtBeginDocument
7273   {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
7274   \ifx\pgfpicture\undefined\else
7275     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
7276     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
7277     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
7278   \fi
7279   \ifx\tikzpicture\undefined\else
7280     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw}%
7281     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
7282     \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
7283     \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw}%
7284   \fi
7285 }}
7286 {}

```

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.

```

7287 \IfBabelLayout{counters*}%
7288   {\bbl@add\bbl@opt@layout{.counters.}%
7289   \directlua{
7290     luatexbase.add_to_callback("process_output_buffer",
7291     Babel.discard_sublr , "Babel.discard_sublr") }%
7292   }{}
7293 \IfBabelLayout{counters}%
7294   {\let\bbl@latinarabic=\@arabic
7295   \let\bbl@0L@\@arabic\@arabic
7296   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7297   \@ifpackagewith{babel}{bidi=default}%

```

```

7298     {\let\bbl@asciroman=\@roman
7299      \let\bbl@OL@roman\@roman
7300      \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7301      \let\bbl@asciiRoman=\@Roman
7302      \let\bbl@OL@Roman\@Roman
7303      \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7304      \let\bbl@OL@labelenumii\labelenumii
7305      \def\labelenumii{}\theenumii}%
7306      \let\bbl@OL@p@enumiii\p@enumiii
7307      \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.

```

7308 \IfBabelLayout{extras}%
7309  {\bbl@carg\let\bbl@OL@underline{underline }%
7310   \bbl@carg\bbl@sreplace{underline }{\hbox}%
7311   {\def\bbl@insidemath{0}\hbox bdir\ifcase\bbl@thetextdir\z@else\@ne\fi}%
7312   \let\bbl@OL@LaTeXe\LaTeXe
7313   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7314     \if b\expandafter\@car\@f@series\@nil\boldmath\fi
7315     \babelsublr{\LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}}
7316   {}
7317 </luatex>

```

10.13 Lua: transforms

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

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

```

7318 <{*transforms>
7319 Babel.linebreaking.replacements = {}
7320 Babel.linebreaking.replacements[0] = {} -- pre
7321 Babel.linebreaking.replacements[1] = {} -- post
7322
7323 function Babel.tovalue(v)
7324   if type(v) == 'table' then
7325     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7326   else
7327     return v
7328   end
7329 end
7330
7331 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7332
7333 function Babel.set_hboxed(head, gc)
7334   for item in node.traverse(head) do
7335     node.set_attribute(item, Babel.attr_hboxed, 1)
7336   end
7337   return head
7338 end
7339
7340 Babel.fetch_subtext = {}
7341
7342 Babel.ignore_pre_char = function(node)
7343   return (node.lang == Babel.nohyphenation)

```

```

7344 end
7345
7346 Babel.show_transforms = false
7347
7348 -- Merging both functions doesn't seem feasible, because there are too
7349 -- many differences.
7350 Babel.fetch_subtext[0] = function(head)
7351   local word_string = ''
7352   local word_nodes = {}
7353   local lang
7354   local item = head
7355   local inmath = false
7356
7357   while item do
7358     if item.id == 11 then
7359       inmath = (item.subtype == 0)
7360     end
7361
7362     if inmath then
7363       -- pass
7364     elseif item.id == 29 then
7365       local locale = node.get_attribute(item, Babel.attr_locale)
7366
7367       if lang == locale or lang == nil then
7368         lang = lang or locale
7369         if Babel.ignore_pre_char(item) then
7370           word_string = word_string .. Babel.us_char
7371         else
7372           if node.has_attribute(item, Babel.attr_hboxed) then
7373             word_string = word_string .. Babel.us_char
7374           else
7375             word_string = word_string .. unicode.utf8.char(item.char)
7376           end
7377         end
7378       end
7379       word_nodes[#word_nodes+1] = item
7380     else
7381       break
7382     end
7383
7384     elseif item.id == 12 and item.subtype == 13 then
7385       if node.has_attribute(item, Babel.attr_hboxed) then
7386         word_string = word_string .. Babel.us_char
7387       else
7388         word_string = word_string .. ' '
7389       end
7390       word_nodes[#word_nodes+1] = item
7391
7392       -- Ignore leading unrecognized nodes, too.
7393       elseif word_string ~= '' then
7394         word_string = word_string .. Babel.us_char
7395         word_nodes[#word_nodes+1] = item -- Will be ignored
7396       end
7397
7398       item = item.next
7399     end
7400
7401     -- Here and above we remove some trailing chars but not the
7402     -- corresponding nodes. But they aren't accessed.
7403     if word_string:sub(-1) == ' ' then
7404       word_string = word_string:sub(1,-2)
7405     end
7406

```

```

7407 if Babel.show_transforms then texio.write_nl(word_string) end
7408 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7409 return word_string, word_nodes, item, lang
7410 end
7411
7412 Babel.fetch_subtext[1] = function(head)
7413   local word_string = ''
7414   local word_nodes = {}
7415   local lang
7416   local item = head
7417   local inmath = false
7418
7419   while item do
7420
7421     if item.id == 11 then
7422       inmath = (item.subtype == 0)
7423     end
7424
7425     if inmath then
7426       -- pass
7427
7428     elseif item.id == 29 then
7429       if item.lang == lang or lang == nil then
7430         lang = lang or item.lang
7431         if node.has_attribute(item, Babel.attr_hboxed) then
7432           word_string = word_string .. Babel.us_char
7433         elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7434           word_string = word_string .. Babel.us_char
7435         else
7436           word_string = word_string .. unicode.utf8.char(item.char)
7437         end
7438         word_nodes[#word_nodes+1] = item
7439       else
7440         break
7441       end
7442
7443     elseif item.id == 7 and item.subtype == 2 then
7444       if node.has_attribute(item, Babel.attr_hboxed) then
7445         word_string = word_string .. Babel.us_char
7446       else
7447         word_string = word_string .. '='
7448       end
7449       word_nodes[#word_nodes+1] = item
7450
7451     elseif item.id == 7 and item.subtype == 3 then
7452       if node.has_attribute(item, Babel.attr_hboxed) then
7453         word_string = word_string .. Babel.us_char
7454       else
7455         word_string = word_string .. '|'
7456       end
7457       word_nodes[#word_nodes+1] = item
7458
7459     -- (1) Go to next word if nothing was found, and (2) implicitly
7460     -- remove leading USs.
7461     elseif word_string == '' then
7462       -- pass
7463
7464     -- This is the responsible for splitting by words.
7465     elseif (item.id == 12 and item.subtype == 13) then
7466       break
7467
7468     else
7469       word_string = word_string .. Babel.us_char

```

```

7470     word_nodes[#word_nodes+1] = item -- Will be ignored
7471 end
7472
7473 item = item.next
7474 end
7475 if Babel.show_transforms then texio.write_nl(word_string) end
7476 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7477 return word_string, word_nodes, item, lang
7478 end
7479
7480 function Babel.pre_hyphenate_replace(head)
7481   Babel.hyphenate_replace(head, 0)
7482 end
7483
7484 function Babel.post_hyphenate_replace(head)
7485   Babel.hyphenate_replace(head, 1)
7486 end
7487
7488 Babel.us_char = string.char(31)
7489
7490 function Babel.hyphenate_replace(head, mode)
7491   local u = unicode.utf8
7492   local lbkr = Babel.linebreaking.replacements[mode]
7493   local tovalue = Babel.tovalue
7494
7495   local word_head = head
7496
7497   if Babel.show_transforms then
7498     texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7499   end
7500
7501   while true do -- for each subtext block
7502
7503     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7504
7505     if Babel.debug then
7506       print()
7507       print((mode == 0) and '@@@<' or '@@@>', w)
7508     end
7509
7510     if nw == nil and w == '' then break end
7511
7512     if not lang then goto next end
7513     if not lbkr[lang] then goto next end
7514
7515     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7516     -- loops are nested.
7517     for k=1, #lbkr[lang] do
7518       local p = lbkr[lang][k].pattern
7519       local r = lbkr[lang][k].replace
7520       local attr = lbkr[lang][k].attr or -1
7521
7522       if Babel.debug then
7523         print('*****', p, mode)
7524       end
7525
7526       -- This variable is set in some cases below to the first *byte*
7527       -- after the match, either as found by u.match (faster) or the
7528       -- computed position based on sc if w has changed.
7529       local last_match = 0
7530       local step = 0
7531
7532       -- For every match.

```

```

7533 while true do
7534     if Babel.debug then
7535         print('====')
7536     end
7537     local new -- used when inserting and removing nodes
7538     local dummy_node -- used by after
7539
7540     local matches = { u.match(w, p, last_match) }
7541
7542     if #matches < 2 then break end
7543
7544     -- Get and remove empty captures (with ()'s, which return a
7545     -- number with the position), and keep actual captures
7546     -- (from (...)), if any, in matches.
7547     local first = table.remove(matches, 1)
7548     local last = table.remove(matches, #matches)
7549     -- Non re-fetched substrings may contain \31, which separates
7550     -- substrings.
7551     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7552
7553     local save_last = last -- with A()BC()D, points to D
7554
7555     -- Fix offsets, from bytes to unicode. Explained above.
7556     first = u.len(w:sub(1, first-1)) + 1
7557     last = u.len(w:sub(1, last-1)) -- now last points to C
7558
7559     -- This loop stores in a small table the nodes
7560     -- corresponding to the pattern. Used by 'data' to provide a
7561     -- predictable behavior with 'insert' (w_nodes is modified on
7562     -- the fly), and also access to 'remove'd nodes.
7563     local sc = first-1 -- Used below, too
7564     local data_nodes = {}
7565
7566     local enabled = true
7567     for q = 1, last-first+1 do
7568         data_nodes[q] = w_nodes[sc+q]
7569         if enabled
7570             and attr > -1
7571             and not node.has_attribute(data_nodes[q], attr)
7572         then
7573             enabled = false
7574         end
7575     end
7576
7577     -- This loop traverses the matched substring and takes the
7578     -- corresponding action stored in the replacement list.
7579     -- sc = the position in substr nodes / string
7580     -- rc = the replacement table index
7581     local rc = 0
7582
7583     ----- TODO. dummy_node?
7584     while rc < last-first+1 or dummy_node do -- for each replacement
7585         if Babel.debug then
7586             print('.....', rc + 1)
7587         end
7588         sc = sc + 1
7589         rc = rc + 1
7590
7591         if Babel.debug then
7592             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7593             local ss = ''
7594             for itt in node.traverse(head) do
7595                 if itt.id == 29 then

```

```

7596         ss = ss .. unicode.utf8.char(itt.char)
7597     else
7598         ss = ss .. '{' .. itt.id .. '}'
7599     end
7600 end
7601 print('*****', ss)
7602
7603 end
7604
7605 local crep = r[rc]
7606 local item = w_nodes[sc]
7607 local item_base = item
7608 local placeholder = Babel.us_char
7609 local d
7610
7611 if crep and crep.data then
7612     item_base = data_nodes[crep.data]
7613 end
7614
7615 if crep then
7616     step = crep.step or step
7617 end
7618
7619 if crep and crep.after then
7620     crep.insert = true
7621     if dummy_node then
7622         item = dummy_node
7623     else -- TODO. if there is a node after?
7624         d = node.copy(item_base)
7625         head, item = node.insert_after(head, item, d)
7626         dummy_node = item
7627     end
7628 end
7629
7630 if crep and not crep.after and dummy_node then
7631     node.remove(head, dummy_node)
7632     dummy_node = nil
7633 end
7634
7635 if not enabled then
7636     last_match = save_last
7637     goto next
7638
7639 elseif crep and next(crep) == nil then -- = {}
7640     if step == 0 then
7641         last_match = save_last -- Optimization
7642     else
7643         last_match = utf8.offset(w, sc+step)
7644     end
7645     goto next
7646
7647 elseif crep == nil or crep.remove then
7648     node.remove(head, item)
7649     table.remove(w_nodes, sc)
7650     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7651     sc = sc - 1 -- Nothing has been inserted.
7652     last_match = utf8.offset(w, sc+1+step)
7653     goto next
7654
7655 elseif crep and crep.kashida then
7656     node.set_attribute(item,
7657         Babel.attr_kashida,
7658         crep.kashida)

```

```

7659         last_match = utf8.offset(w, sc+1+step)
7660         goto next
7661
7662     elseif crep and crep.string then
7663         local str = crep.string(matches)
7664         if str == '' then -- Gather with nil
7665             node.remove(head, item)
7666             table.remove(w_nodes, sc)
7667             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7668             sc = sc - 1 -- Nothing has been inserted.
7669         else
7670             local loop_first = true
7671             for s in string.utfvalues(str) do
7672                 d = node.copy(item_base)
7673                 d.char = s
7674                 if loop_first then
7675                     loop_first = false
7676                     head, new = node.insert_before(head, item, d)
7677                     if sc == 1 then
7678                         word_head = head
7679                     end
7680                     w_nodes[sc] = d
7681                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7682                 else
7683                     sc = sc + 1
7684                     head, new = node.insert_before(head, item, d)
7685                     table.insert(w_nodes, sc, new)
7686                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7687                 end
7688                 if Babel.debug then
7689                     print('.....', 'str')
7690                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7691                 end
7692             end -- for
7693             node.remove(head, item)
7694         end -- if ''
7695         last_match = utf8.offset(w, sc+1+step)
7696         goto next
7697
7698     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7699         d = node.new(7, 3) -- (disc, regular)
7700         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7701         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7702         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7703         d.attr = item_base.attr
7704         if crep.pre == nil then -- TeXbook p96
7705             d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7706         else
7707             d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7708         end
7709         placeholder = '|'
7710         head, new = node.insert_before(head, item, d)
7711
7712     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7713         -- ERROR
7714
7715     elseif crep and crep.penalty then
7716         d = node.new(14, 0) -- (penalty, userpenalty)
7717         d.attr = item_base.attr
7718         d.penalty = tovalue(crep.penalty)
7719         head, new = node.insert_before(head, item, d)
7720
7721     elseif crep and crep.space then

```

```

7722         -- 655360 = 10 pt = 10 * 65536 sp
7723         d = node.new(12, 13)          -- (glue, spaceskip)
7724         local quad = font.getfont(item_base.font).size or 655360
7725         node.setglue(d, tovalue(crep.space[1]) * quad,
7726                      tovalue(crep.space[2]) * quad,
7727                      tovalue(crep.space[3]) * quad)
7728         if mode == 0 then
7729             placeholder = ' '
7730         end
7731         head, new = node.insert_before(head, item, d)
7732
7733     elseif crep and crep.norule then
7734         -- 655360 = 10 pt = 10 * 65536 sp
7735         d = node.new(2, 3)             -- (rule, empty) = \no*rule
7736         local quad = font.getfont(item_base.font).size or 655360
7737         d.width  = tovalue(crep.norule[1]) * quad
7738         d.height = tovalue(crep.norule[2]) * quad
7739         d.depth  = tovalue(crep.norule[3]) * quad
7740         head, new = node.insert_before(head, item, d)
7741
7742     elseif crep and crep.spacefactor then
7743         d = node.new(12, 13)          -- (glue, spaceskip)
7744         local base_font = font.getfont(item_base.font)
7745         node.setglue(d,
7746                     tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7747                     tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7748                     tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7749         if mode == 0 then
7750             placeholder = ' '
7751         end
7752         head, new = node.insert_before(head, item, d)
7753
7754     elseif mode == 0 and crep and crep.space then
7755         -- ERROR
7756
7757     elseif crep and crep.kern then
7758         d = node.new(13, 1)           -- (kern, user)
7759         local quad = font.getfont(item_base.font).size or 655360
7760         d.attr = item_base.attr
7761         d.kern = tovalue(crep.kern) * quad
7762         head, new = node.insert_before(head, item, d)
7763
7764     elseif crep and crep.node then
7765         d = node.new(crep.node[1], crep.node[2])
7766         d.attr = item_base.attr
7767         head, new = node.insert_before(head, item, d)
7768
7769     end -- i.e., replacement cases
7770
7771     -- Shared by disc, space(factor), kern, node and penalty.
7772     if sc == 1 then
7773         word_head = head
7774     end
7775     if crep.insert then
7776         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7777         table.insert(w_nodes, sc, new)
7778         last = last + 1
7779     else
7780         w_nodes[sc] = d
7781         node.remove(head, item)
7782         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7783     end
7784

```

```

7785         last_match = utf8.offset(w, sc+1+step)
7786
7787         ::next::
7788
7789     end -- for each replacement
7790
7791     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7792     if Babel.debug then
7793         print('.....', '/')
7794         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7795     end
7796
7797     if dummy_node then
7798         node.remove(head, dummy_node)
7799         dummy_node = nil
7800     end
7801
7802     end -- for match
7803
7804 end -- for patterns
7805
7806 ::next::
7807 word_head = nw
7808 end -- for substring
7809
7810 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7811 return head
7812 end
7813
7814 -- This table stores capture maps, numbered consecutively
7815 Babel.capture_maps = {}
7816
7817 function Babel.esc_hex_to_char(h)
7818     if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7819         tex.getcatcode(tonumber(h, 16)) ~= 12 then
7820         return string.format([[\Uchar"%X ]], tonumber(h,16))
7821     else
7822         return unicode.utf8.char(tonumber(h, 16))
7823     end
7824 end
7825
7826 -- The following functions belong to the next macro
7827 function Babel.capture_func(key, cap)
7828     local ret = "[[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[[[" .. "]"
7829     local cnt
7830     local u = unicode.utf8
7831     ret = u.gsub(ret, '{(%x%x%x%x+)}', '\x01%\x04')
7832     ret, cnt = ret:gsub('{{[0-9]}|([^\^]|+)|(.-)}', Babel.capture_func_map)
7833     ret = u.gsub(ret, '\x01(%x%x%x%x+)\x04', Babel.esc_hex_to_char)
7834     ret = ret:gsub("%[%[%]%]%.%", '')
7835     ret = ret:gsub("%.%[%[%]%]", '')
7836     return key .. "[[=function(m) return ]] .. ret .. [[ end]]
7837 end
7838
7839 function Babel.capt_map(from, mapno)
7840     return Babel.capture_maps[mapno][from] or from
7841 end
7842
7843 -- Handle the {n|abc|ABC} syntax in captures
7844 function Babel.capture_func_map(capno, from, to)
7845     local u = unicode.utf8
7846     from = u.gsub(from, '\x01(%x%x%x%x+)\x04',
7847         function (n)

```

```

7848         return u.char(tonumber(n, 16))
7849     end)
7850 to = u.gsub(to, '\\x01(%x%x%x%x+)\x04',
7851     function (n)
7852         return u.char(tonumber(n, 16))
7853     end)
7854 local froms = {}
7855 for s in string.utfcharacters(from) do
7856     table.insert(froms, s)
7857 end
7858 local cnt = 1
7859 table.insert(Babel.capture_maps, {})
7860 local mlen = table.getn(Babel.capture_maps)
7861 for s in string.utfcharacters(to) do
7862     Babel.capture_maps[mlen][froms[cnt]] = s
7863     cnt = cnt + 1
7864 end
7865 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7866     (mlen) .. ").." .. "["
7867 end
7868
7869 -- Create/Extend reversed sorted list of kashida weights:
7870 function Babel.capture_kashida(key, wt)
7871     wt = tonumber(wt)
7872     if Babel.kashida_wts then
7873         for p, q in ipairs(Babel.kashida_wts) do
7874             if wt == q then
7875                 break
7876             elseif wt > q then
7877                 table.insert(Babel.kashida_wts, p, wt)
7878                 break
7879             elseif table.getn(Babel.kashida_wts) == p then
7880                 table.insert(Babel.kashida_wts, wt)
7881             end
7882         end
7883     else
7884         Babel.kashida_wts = { wt }
7885     end
7886     return 'kashida = ' .. wt
7887 end
7888
7889 function Babel.capture_node(id, subtype)
7890     local sbt = 0
7891     for k, v in pairs(node.subtypes(id)) do
7892         if v == subtype then sbt = k end
7893     end
7894     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7895 end
7896
7897 -- Experimental: applies prehyphenation transforms to a string (letters
7898 -- and spaces).
7899 function Babel.string_prehyphenation(str, locale)
7900     local n, head, last, res
7901     head = node.new(8, 0) -- dummy (hack just to start)
7902     last = head
7903     for s in string.utfvalues(str) do
7904         if s == 20 then
7905             n = node.new(12, 0)
7906         else
7907             n = node.new(29, 0)
7908             n.char = s
7909         end
7910         node.set_attribute(n, Babel.attr_locale, locale)

```

```

7911     last.next = n
7912     last = n
7913 end
7914 head = Babel.hyphenate_replace(head, 0)
7915 res = ''
7916 for n in node.traverse(head) do
7917     if n.id == 12 then
7918         res = res .. ' '
7919     elseif n.id == 29 then
7920         res = res .. unicode.utf8.char(n.char)
7921     end
7922 end
7923 tex.print(res)
7924 end
7925 </transforms>

```

10.14 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

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

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

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

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

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

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

```

7926 <(*basic-r>
7927 Babel.bidi_enabled = true
7928
7929 require('babel-data-bidi.lua')
7930
7931 local characters = Babel.characters
7932 local ranges = Babel.ranges
7933

```

```

7934 local DIR = node.id("dir")
7935
7936 local function dir_mark(head, from, to, outer)
7937   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7938   local d = node.new(DIR)
7939   d.dir = '+' .. dir
7940   node.insert_before(head, from, d)
7941   d = node.new(DIR)
7942   d.dir = '-' .. dir
7943   node.insert_after(head, to, d)
7944 end
7945
7946 function Babel.bidi(head, ispar)
7947   local first_n, last_n          -- first and last char with nums
7948   local last_es                  -- an auxiliary 'last' used with nums
7949   local first_d, last_d          -- first and last char in L/R block
7950   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):

```

7951   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7952   local strong_lr = (strong == 'l') and 'l' or 'r'
7953   local outer = strong
7954
7955   local new_dir = false
7956   local first_dir = false
7957   local inmath = false
7958
7959   local last_lr
7960
7961   local type_n = ''
7962
7963   for item in node.traverse(head) do
7964
7965     -- three cases: glyph, dir, otherwise
7966     if item.id == node.id'glyph'
7967       or (item.id == 7 and item.subtype == 2) then
7968
7969       local itemchar
7970       if item.id == 7 and item.subtype == 2 then
7971         itemchar = item.replace.char
7972       else
7973         itemchar = item.char
7974       end
7975       local chardata = characters[itemchar]
7976       dir = chardata and chardata.d or nil
7977       if not dir then
7978         for nn, et in ipairs(ranges) do
7979           if itemchar < et[1] then
7980             break
7981           elseif itemchar <= et[2] then
7982             dir = et[3]
7983             break
7984           end
7985         end
7986       end
7987       dir = dir or 'l'
7988       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).

```

7989     if new_dir then
7990         attr_dir = 0
7991         for at in node.traverse(item.attr) do
7992             if at.number == Babel.attr_dir then
7993                 attr_dir = at.value & 0x3
7994             end
7995         end
7996         if attr_dir == 1 then
7997             strong = 'r'
7998         elseif attr_dir == 2 then
7999             strong = 'al'
8000         else
8001             strong = 'l'
8002         end
8003         strong_lr = (strong == 'l') and 'l' or 'r'
8004         outer = strong_lr
8005         new_dir = false
8006     end
8007
8008     if dir == 'nsm' then dir = strong end          -- W1

```

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

```

8009     dir_real = dir          -- We need dir_real to set strong below
8010     if dir == 'al' then dir = 'r' end -- W3

```

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

```

8011     if strong == 'al' then
8012         if dir == 'en' then dir = 'an' end          -- W2
8013         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
8014         strong_lr = 'r'          -- W3
8015     end

```

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

```

8016     elseif item.id == node.id'dir' and not inmath then
8017         new_dir = true
8018         dir = nil
8019     elseif item.id == node.id'math' then
8020         inmath = (item.subtype == 0)
8021     else
8022         dir = nil          -- Not a char
8023     end

```

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

```

8024     if dir == 'en' or dir == 'an' or dir == 'et' then
8025         if dir ~= 'et' then
8026             type_n = dir
8027         end
8028         first_n = first_n or item
8029         last_n = last_es or item
8030         last_es = nil
8031     elseif dir == 'es' and last_n then -- W3+W6
8032         last_es = item
8033     elseif dir == 'cs' then          -- it's right - do nothing
8034     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
8035         if strong_lr == 'r' and type_n ~= '' then
8036             dir_mark(head, first_n, last_n, 'r')
8037         elseif strong_lr == 'l' and first_d and type_n == 'an' then

```

```

8038     dir_mark(head, first_n, last_n, 'r')
8039     dir_mark(head, first_d, last_d, outer)
8040     first_d, last_d = nil, nil
8041     elseif strong_lr == 'l' and type_n ~= '' then
8042         last_d = last_n
8043     end
8044     type_n = ''
8045     first_n, last_n = nil, nil
8046 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:

```

8047 if dir == 'l' or dir == 'r' then
8048     if dir ~= outer then
8049         first_d = first_d or item
8050         last_d = item
8051     elseif first_d and dir ~= strong_lr then
8052         dir_mark(head, first_d, last_d, outer)
8053         first_d, last_d = nil, nil
8054     end
8055 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.

```

8056 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
8057     item.char = characters[item.char] and
8058         characters[item.char].m or item.char
8059 elseif (dir or new_dir) and last_lr ~= item then
8060     local mir = outer .. strong_lr .. (dir or outer)
8061     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
8062         for ch in node.traverse(node.next(last_lr)) do
8063             if ch == item then break end
8064             if ch.id == node.id'glyph' and characters[ch.char] then
8065                 ch.char = characters[ch.char].m or ch.char
8066             end
8067         end
8068     end
8069 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).

```

8070 if dir == 'l' or dir == 'r' then
8071     last_lr = item
8072     strong = dir_real -- Don't search back - best save now
8073     strong_lr = (strong == 'l') and 'l' or 'r'
8074 elseif new_dir then
8075     last_lr = nil
8076 end
8077 end

```

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

```

8078 if last_lr and outer == 'r' then
8079     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
8080         if characters[ch.char] then
8081             ch.char = characters[ch.char].m or ch.char
8082         end
8083     end
8084 end
8085 if first_n then

```

```

8086     dir_mark(head, first_n, last_n, outer)
8087 end
8088 if first_d then
8089     dir_mark(head, first_d, last_d, outer)
8090 end

```

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

```

8091 return node.prev(head) or head
8092 end
8093 </basic-r>

```

And here the Lua code for bidi=basic:

```

8094 (*basic)
8095 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
8096
8097 Babel.fontmap = Babel.fontmap or {}
8098 Babel.fontmap[0] = {}      -- l
8099 Babel.fontmap[1] = {}      -- r
8100 Babel.fontmap[2] = {}      -- al/an
8101
8102 -- To cancel mirroring. Also OML, OMS, U?
8103 Babel.symbol_fonts = Babel.symbol_fonts or {}
8104 Babel.symbol_fonts[font.id('tenln')] = true
8105 Babel.symbol_fonts[font.id('tenlnw')] = true
8106 Babel.symbol_fonts[font.id('tencirc')] = true
8107 Babel.symbol_fonts[font.id('tencircw')] = true
8108
8109 Babel.bidi_enabled = true
8110 Babel.mirroring_enabled = true
8111
8112 require('babel-data-bidi.lua')
8113
8114 local characters = Babel.characters
8115 local ranges = Babel.ranges
8116
8117 local DIR = node.id('dir')
8118 local GLYPH = node.id('glyph')
8119
8120 local function insert_implicit(head, state, outer)
8121     local new_state = state
8122     if state.sim and state.eim and state.sim ~= state.eim then
8123         dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
8124         local d = node.new(DIR)
8125         d.dir = '+' .. dir
8126         node.insert_before(head, state.sim, d)
8127         local d = node.new(DIR)
8128         d.dir = '-' .. dir
8129         node.insert_after(head, state.eim, d)
8130     end
8131     new_state.sim, new_state.eim = nil, nil
8132     return head, new_state
8133 end
8134
8135 local function insert_numeric(head, state)
8136     local new
8137     local new_state = state
8138     if state.san and state.ean and state.san ~= state.ean then
8139         local d = node.new(DIR)
8140         d.dir = '+TLT'
8141         _, new = node.insert_before(head, state.san, d)
8142         if state.san == state.sim then state.sim = new end
8143         local d = node.new(DIR)
8144         d.dir = '-TLT'

```

```

8145     _, new = node.insert_after(head, state.ean, d)
8146     if state.ean == state.eim then state.eim = new end
8147 end
8148 new_state.san, new_state.ean = nil, nil
8149 return head, new_state
8150 end
8151
8152 local function glyph_not_symbol_font(node)
8153   if node.id == GLYPH then
8154     return not Babel.symbol_fonts[node.font]
8155   else
8156     return false
8157   end
8158 end
8159
8160 -- TODO - \hbox with an explicit dir can lead to wrong results
8161 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
8162 -- was made to improve the situation, but the problem is the 3-dir
8163 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8164 -- well.
8165
8166 function Babel.bidi(head, ispar, hdir)
8167   local d    -- d is used mainly for computations in a loop
8168   local prev_d = ''
8169   local new_d = false
8170
8171   local nodes = {}
8172   local outer_first = nil
8173   local inmath = false
8174
8175   local glue_d = nil
8176   local glue_i = nil
8177
8178   local has_en = false
8179   local first_et = nil
8180
8181   local has_hyperlink = false
8182
8183   local ATDIR = Babel.attr_dir
8184   local attr_d, temp
8185   local locale_d
8186
8187   local save_outer
8188   local locale_d = node.get_attribute(head, ATDIR)
8189   if locale_d then
8190     locale_d = locale_d & 0x3
8191     save_outer = (locale_d == 0 and 'l') or
8192                 (locale_d == 1 and 'r') or
8193                 (locale_d == 2 and 'al')
8194   elseif ispar then -- Or error? Shouldn't happen
8195     -- when the callback is called, we are just _after_ the box,
8196     -- and the textdir is that of the surrounding text
8197     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8198   else -- Empty box
8199     save_outer = ('TRT' == hdir) and 'r' or 'l'
8200   end
8201   local outer = save_outer
8202   local last = outer
8203   -- 'al' is only taken into account in the first, current loop
8204   if save_outer == 'al' then save_outer = 'r' end
8205
8206   local fontmap = Babel.fontmap
8207

```

```

8208 for item in node.traverse(head) do
8209
8210     -- Mask: DxxxPPTT (Done, Paddir [0-2], Textdir [0-2])
8211     locale_d = node.get_attribute(item, ATDIR)
8212     node.set_attribute(item, ATDIR, 0x80)
8213
8214     -- In what follows, #node is the last (previous) node, because the
8215     -- current one is not added until we start processing the neutrals.
8216     -- three cases: glyph, dir, otherwise
8217     if glyph_not_symbol_font(item)
8218         or (item.id == 7 and item.subtype == 2) then
8219
8220         if inmath then goto nextnode end
8221
8222         if locale_d == 0x80 then goto nextnode end
8223         locale_d = locale_d or ((save_outer=='l') and 0 or 1)
8224
8225         local d_font = nil
8226         local item_r
8227         if item.id == 7 and item.subtype == 2 then
8228             item_r = item.replace    -- automatic discs have just 1 glyph
8229         else
8230             item_r = item
8231         end
8232
8233         local chardata = characters[item_r.char]
8234         d = chardata and chardata.d or nil
8235         if not d or d == 'nsm' then
8236             for nn, et in ipairs(ranges) do
8237                 if item_r.char < et[1] then
8238                     break
8239                 elseif item_r.char <= et[2] then
8240                     if not d then d = et[3]
8241                     elseif d == 'nsm' then d_font = et[3]
8242                     end
8243                     break
8244                 end
8245             end
8246         end
8247         d = d or 'l'
8248
8249         -- A short 'pause' in bidi for mapfont
8250         -- %%% TODO. move if fontmap here
8251         d_font = d_font or d
8252         d_font = (d_font == 'l' and 0) or
8253                 (d_font == 'nsm' and 0) or
8254                 (d_font == 'r' and 1) or
8255                 (d_font == 'al' and 2) or
8256                 (d_font == 'an' and 2) or nil
8257         if d_font and fontmap and fontmap[d_font][item_r.font] then
8258             item_r.font = fontmap[d_font][item_r.font]
8259         end
8260
8261         if new_d then
8262             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8263             if inmath then
8264                 attr_d = 0
8265             else
8266                 attr_d = locale_d & 0x3
8267             end
8268             if attr_d == 1 then
8269                 outer_first = 'r'
8270                 last = 'r'

```

```

8271         elseif attr_d == 2 then
8272             outer_first = 'r'
8273             last = 'al'
8274         else
8275             outer_first = 'l'
8276             last = 'l'
8277         end
8278         outer = last
8279         has_en = false
8280         first_et = nil
8281         new_d = false
8282     end
8283
8284     if glue_d then
8285         if (d == 'l' and 'l' or 'r') ~= glue_d then
8286             table.insert(nodes, {glue_i, 'on', nil})
8287         end
8288         glue_d = nil
8289         glue_i = nil
8290     end
8291
8292     elseif item.id == DIR then
8293         if inmath then goto nextnode end
8294         d = nil
8295         new_d = true
8296
8297     elseif item.id == node.id'glue' and item.subtype == 13 then
8298         if inmath then goto nextnode end
8299         glue_d = d
8300         glue_i = item
8301         d = nil
8302
8303     elseif item.id == node.id'math' then
8304         if item.subtype == 0 then -- mathon
8305             inmath = true
8306             d = 'on'
8307             table.insert(nodes, {item, 'on', outer_first})
8308             outer_first = nil
8309             goto nextnode
8310         else -- mathoff
8311             inmath = false
8312         end
8313
8314     elseif item.id == 8 and item.subtype == 19 then
8315         has_hyperlink = true
8316
8317     else
8318         d = nil
8319     end
8320
8321     -- AL <= EN/ET/ES      -- W2 + W3 + W6
8322     if last == 'al' and d == 'en' then
8323         d = 'an'          -- W3
8324     elseif last == 'al' and (d == 'et' or d == 'es') then
8325         d = 'on'          -- W6
8326     end
8327
8328     -- EN + CS/ES + EN      -- W4
8329     if d == 'en' and #nodes >= 2 then
8330         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8331             and nodes[#nodes-1][2] == 'en' then
8332             nodes[#nodes][2] = 'en'
8333         end

```

```

8334     end
8335
8336     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8337     if d == 'an' and #nodes >= 2 then
8338         if (nodes[#nodes][2] == 'cs')
8339             and nodes[#nodes-1][2] == 'an' then
8340             nodes[#nodes][2] = 'an'
8341         end
8342     end
8343
8344     -- ET/EN                  -- W5 + W7->l / W6->on
8345     if d == 'et' then
8346         first_et = first_et or (#nodes + 1)
8347     elseif d == 'en' then
8348         has_en = true
8349         first_et = first_et or (#nodes + 1)
8350     elseif first_et then      -- d may be nil here !
8351         if has_en then
8352             if last == 'l' then
8353                 temp = 'l'    -- W7
8354             else
8355                 temp = 'en'   -- W5
8356             end
8357         else
8358             temp = 'on'       -- W6
8359         end
8360         for e = first_et, #nodes do
8361             if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8362         end
8363         first_et = nil
8364         has_en = false
8365     end
8366
8367     -- Force mathdir in math if ON (currently works as expected only
8368     -- with 'l')
8369
8370     if inmath and d == 'on' then
8371         d = ('TRT' == tex.mathdir) and 'r' or 'l'
8372     end
8373
8374     if d then
8375         if d == 'al' then
8376             d = 'r'
8377             last = 'al'
8378         elseif d == 'l' or d == 'r' then
8379             last = d
8380         end
8381         prev_d = d
8382         table.insert(nodes, {item, d, outer_first})
8383     end
8384
8385     outer_first = nil
8386
8387     ::nextnode::
8388
8389 end -- for each node
8390
8391 -- TODO -- repeated here in case EN/ET is the last node. Find a
8392 -- better way of doing things:
8393 if first_et then      -- dir may be nil here !
8394     if has_en then
8395         if last == 'l' then
8396             temp = 'l'    -- W7

```

```

8397     else
8398         temp = 'en'    -- W5
8399     end
8400 else
8401     temp = 'on'        -- W6
8402 end
8403 for e = first_et, #nodes do
8404     if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8405 end
8406 end
8407
8408 -- dummy node, to close things
8409 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8410
8411 ----- NEUTRAL -----
8412
8413 outer = save_outer
8414 last = outer
8415
8416 local first_on = nil
8417
8418 for q = 1, #nodes do
8419     local item
8420
8421     local outer_first = nodes[q][3]
8422     outer = outer_first or outer
8423     last = outer_first or last
8424
8425     local d = nodes[q][2]
8426     if d == 'an' or d == 'en' then d = 'r' end
8427     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8428
8429     if d == 'on' then
8430         first_on = first_on or q
8431     elseif first_on then
8432         if last == d then
8433             temp = d
8434         else
8435             temp = outer
8436         end
8437         for r = first_on, q - 1 do
8438             nodes[r][2] = temp
8439             item = nodes[r][1]    -- MIRRORING
8440             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8441                 and temp == 'r' and characters[item.char] then
8442                 local font_mode = ''
8443                 if item.font > 0 and font.fonts[item.font].properties then
8444                     font_mode = font.fonts[item.font].properties.mode
8445                 end
8446                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8447                     item.char = characters[item.char].m or item.char
8448                 end
8449             end
8450         end
8451         first_on = nil
8452     end
8453
8454     if d == 'r' or d == 'l' then last = d end
8455 end
8456
8457 ----- IMPLICIT, REORDER -----
8458
8459 outer = save_outer

```

```

8460 last = outer
8461
8462 local state = {}
8463 state.has_r = false
8464
8465 for q = 1, #nodes do
8466
8467     local item = nodes[q][1]
8468
8469     outer = nodes[q][3] or outer
8470
8471     local d = nodes[q][2]
8472
8473     if d == 'nsm' then d = last end          -- W1
8474     if d == 'en' then d = 'an' end
8475     local isdir = (d == 'r' or d == 'l')
8476
8477     if outer == 'l' and d == 'an' then
8478         state.san = state.san or item
8479         state.ean = item
8480     elseif state.san then
8481         head, state = insert_numeric(head, state)
8482     end
8483
8484     if outer == 'l' then
8485         if d == 'an' or d == 'r' then      -- im -> implicit
8486             if d == 'r' then state.has_r = true end
8487             state.sim = state.sim or item
8488             state.eim = item
8489         elseif d == 'l' and state.sim and state.has_r then
8490             head, state = insert_implicit(head, state, outer)
8491         elseif d == 'l' then
8492             state.sim, state.eim, state.has_r = nil, nil, false
8493         end
8494     else
8495         if d == 'an' or d == 'l' then
8496             if nodes[q][3] then -- nil except after an explicit dir
8497                 state.sim = item -- so we move sim 'inside' the group
8498             else
8499                 state.sim = state.sim or item
8500             end
8501             state.eim = item
8502         elseif d == 'r' and state.sim then
8503             head, state = insert_implicit(head, state, outer)
8504         elseif d == 'r' then
8505             state.sim, state.eim = nil, nil
8506         end
8507     end
8508
8509     if isdir then
8510         last = d          -- Don't search back - best save now
8511     elseif d == 'on' and state.san then
8512         state.san = state.san or item
8513         state.ean = item
8514     end
8515
8516 end
8517
8518 head = node.prev(head) or head
8519 % \end{macrocode}
8520 %
8521 % Now direction nodes has been distributed with relation to characters
8522 % and spaces, we need to take into account \TeX-specific elements in

```

```

8523% the node list, to move them at an appropriate place. Firstly, with
8524% hyperlinks. Secondly, we avoid them between penalties and spaces, so
8525% that the latter are still discardable.
8526%
8527% \begin{macrocode}
8528 --- FIXES ---
8529 if has_hyperlink then
8530   local flag, linking = 0, 0
8531   for item in node.traverse(head) do
8532     if item.id == DIR then
8533       if item.dir == '+TRT' or item.dir == '+TLT' then
8534         flag = flag + 1
8535       elseif item.dir == '-TRT' or item.dir == '-TLT' then
8536         flag = flag - 1
8537       end
8538     elseif item.id == 8 and item.subtype == 19 then
8539       linking = flag
8540     elseif item.id == 8 and item.subtype == 20 then
8541       if linking > 0 then
8542         if item.prev.id == DIR and
8543            (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8544           d = node.new(DIR)
8545           d.dir = item.prev.dir
8546           node.remove(head, item.prev)
8547           node.insert_after(head, item, d)
8548         end
8549       end
8550       linking = 0
8551     end
8552   end
8553 end
8554
8555 for item in node.traverse_id(10, head) do
8556   local p = item
8557   local flag = false
8558   while p.prev and p.prev.id == 14 do
8559     flag = true
8560     p = p.prev
8561   end
8562   if flag then
8563     node.insert_before(head, p, node.copy(item))
8564     node.remove(head, item)
8565   end
8566 end
8567
8568 return head
8569 end
8570
8571 function Babel.unset_atdir(head)
8572   local ATDIR = Babel.attr_dir
8573   for item in node.traverse(head) do
8574     node.set_attribute(item, ATDIR, 0x80)
8575   end
8576   return head
8577 end
8578 \end{macrocode}
8579 \end{basic}

```

11. Data for CJK

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

```
% [0x0021]={c='ex'},
% [0x0024]={c='pr'},
% [0x0025]={c='po'},
% [0x0028]={c='op'},
% [0x0029]={c='cp'},
% [0x002B]={c='pr'},
%
```

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

12. The ‘nil’ language

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

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

```
8578 <{*nil}
8579 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8580 \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.

```
8581 \ifx\l@nil\undefined
8582 \newlanguage\l@nil
8583 \@namedef{bbl@hyphendata@the\l@nil}{}{}{}% Remove warning
8584 \let\bbl@elt\relax
8585 \edef\bbl@languages{% Add it to the list of languages
8586 \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
8587 \fi
```

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

```
8588 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

`\captionnil`
`\datenil`

```
8589 \let\captionnil\@empty
8590 \let\datenil\@empty
```

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

```
8591 \def\bbl@inidata@nil{%
8592 \bbl@elt{identification}{tag.ini}{und}%
8593 \bbl@elt{identification}{load.level}{0}%
8594 \bbl@elt{identification}{charset}{utf8}%
8595 \bbl@elt{identification}{version}{1.0}%
8596 \bbl@elt{identification}{date}{2022-05-16}%
8597 \bbl@elt{identification}{name.local}{nil}%
8598 \bbl@elt{identification}{name.english}{nil}%
8599 \bbl@elt{identification}{name.babel}{nil}%
8600 \bbl@elt{identification}{tag.bcp47}{und}%
8601 \bbl@elt{identification}{language.tag.bcp47}{und}%
8602 \bbl@elt{identification}{tag.opentype}{dflt}%
8603 \bbl@elt{identification}{script.name}{Latin}%
8604 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8605 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8606 \bbl@elt{identification}{level}{1}%
8607 \bbl@elt{identification}{encodings}{}%
8608 \bbl@elt{identification}{derivate}{no}}
8609 \@namedef{bbl@tbc@nil}{und}
8610 \@namedef{bbl@lbc@nil}{und}
```

```

8611 \@namedef{bbl@casing@nil}{und}
8612 \@namedef{bbl@lotf@nil}{dflt}
8613 \@namedef{bbl@elname@nil}{nil}
8614 \@namedef{bbl@lname@nil}{nil}
8615 \@namedef{bbl@esname@nil}{Latin}
8616 \@namedef{bbl@sname@nil}{Latin}
8617 \@namedef{bbl@sbc@nil}{Latn}
8618 \@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.

```

8619 \ldf@finish{nil}
8620 </nil>

```

13. Calendars

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

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

```

8621 <<*Compute Julian day>> ≡
8622 \def\bbl@fpmmod#1#2{(#1-#2*floor((#1)/(#2)))}
8623 \def\bbl@cs@gregleap#1{%
8624   (\bbl@fpmmod{#1}{4} == 0) &&
8625   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8626 \def\bbl@cs@jd#1#2#3{% year, month, day
8627   \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8628     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8629     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8630     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8631 <</Compute Julian day>>

```

13.1. Islamic

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

```

8632 <*ca-islamic>
8633 <@Compute Julian day@>
8634 % == islamic (default)
8635 % Not yet implemented
8636 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8637 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8638   ((#3 + ceil(29.5 * (#2 - 1)) +
8639     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8640     1948439.5) - 1) }
8641 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8642 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8643 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8644 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8645 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8646 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8647   \edef\bbl@tempa{%
8648     \fpeval{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 } }%
8649   \edef#5{%
8650     \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) } }%
8651   \edef#6{\fpeval{
8652     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) } }%
8653   \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).

```

8654 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8655 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8656 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8657 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8658 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8659 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8660 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8661 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8662 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8663 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8664 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8665 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8666 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8667 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8668 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8669 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8670 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8671 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8672 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8673 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8674 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8675 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8676 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8677 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8678 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8679 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8680 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8681 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8682 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8683 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8684 65401,65431,65460,65490,65520}
8685 \@namedef\bbl@ca@islamic-umalqura+{\bbl@ca@islamcuqr@x{+1}}
8686 \@namedef\bbl@ca@islamic-umalqura-{\bbl@ca@islamcuqr@x{-1}}
8687 \@namedef\bbl@ca@islamic-umalqura-{\bbl@ca@islamcuqr@x{-1}}
8688 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8689 \ifnum#2>2014 \ifnum#2<2038
8690 \bbl@afterfi\expandafter\@gobble
8691 \fi\fi
8692 {\bbl@error{year-out-range}{2014-2038}{}}}%
8693 \edef\bbl@tempd{\fpeval{ % (Julian) day
8694 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8695 \count@\@ne
8696 \bbl@foreach\bbl@cs@umalqura@data{%
8697 \advance\count@\@ne
8698 \ifnum##1>\bbl@tempd\else
8699 \edef\bbl@tempe{\the\count@}%
8700 \edef\bbl@tempb{##1}%
8701 \fi}%
8702 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8703 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1 ) / 12) }}% annus
8704 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8705 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8706 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8707 \bbl@add\bbl@precalendar{%
8708 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8709 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8710 \bbl@replace\bbl@ld@calendar{+}{}}%
8711 \bbl@replace\bbl@ld@calendar{-}{}}%
8712 </ca-islamic>

```

13.2. Hebrew

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

```
8713 (*ca-hebrew)
8714 \newcount\bbl@cntcommon
8715 \def\bbl@remainder#1#2#3{%
8716   #3=#1\relax
8717   \divide #3 by #2\relax
8718   \multiply #3 by -#2\relax
8719   \advance #3 by #1\relax}%
8720 \newif\ifbbl@divisible
8721 \def\bbl@checkifdivisible#1#2{%
8722   {\countdef\tmp=0
8723     \bbl@remainder{#1}{#2}{\tmp}%
8724     \ifnum \tmp=0
8725       \global\bbl@divisibletrue
8726     \else
8727       \global\bbl@divisiblefalse
8728     \fi}}
8729 \newif\ifbbl@gregleap
8730 \def\bbl@ifgregleap#1{%
8731   \bbl@checkifdivisible{#1}{4}%
8732   \ifbbl@divisible
8733     \bbl@checkifdivisible{#1}{100}%
8734     \ifbbl@divisible
8735       \bbl@checkifdivisible{#1}{400}%
8736       \ifbbl@divisible
8737         \bbl@gregleaptrue
8738       \else
8739         \bbl@gregleapfalse
8740       \fi
8741     \else
8742       \bbl@gregleaptrue
8743     \fi
8744   \else
8745     \bbl@gregleapfalse
8746   \fi
8747   \ifbbl@gregleap}
8748 \def\bbl@gregdayspriormonths#1#2#3{%
8749   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8750     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8751   \bbl@ifgregleap{#2}%
8752   \ifnum #1 > 2
8753     \advance #3 by 1
8754   \fi
8755   \fi
8756   \global\bbl@cntcommon=#3}%
8757   #3=\bbl@cntcommon}
8758 \def\bbl@gregdaysprioryears#1#2{%
8759   {\countdef\tmpc=4
8760     \countdef\tmpb=2
8761     \tmpb=#1\relax
8762     \advance \tmpb by -1
8763     \tmpc=\tmpb
8764     \multiply \tmpc by 365
8765     #2=\tmpc
8766     \tmpc=\tmpb
8767     \divide \tmpc by 4
8768     \advance #2 by \tmpc
8769     \tmpc=\tmpb
8770     \divide \tmpc by 100
```

```

8771 \advance #2 by -\tmpc
8772 \tmpc=\tmpb
8773 \divide \tmpc by 400
8774 \advance #2 by \tmpc
8775 \global\bbl@cntcommon=#2\relax}%
8776 #2=\bbl@cntcommon}
8777 \def\bbl@absfromgreg#1#2#3#4{%
8778 {\countdef\tmpd=0
8779 #4=#1\relax
8780 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8781 \advance #4 by \tmpd
8782 \bbl@gregdaysprioryears{#3}{\tmpd}%
8783 \advance #4 by \tmpd
8784 \global\bbl@cntcommon=#4\relax}%
8785 #4=\bbl@cntcommon}
8786 \newif\ifbbl@hebrleap
8787 \def\bbl@checkleaphebryear#1{%
8788 {\countdef\tmpa=0
8789 \countdef\tmpb=1
8790 \tmpa=#1\relax
8791 \multiply \tmpa by 7
8792 \advance \tmpa by 1
8793 \bbl@remainder{\tmpa}{19}{\tmpb}%
8794 \ifnum \tmpb < 7
8795 \global\bbl@hebrleaptrue
8796 \else
8797 \global\bbl@hebrleapfalse
8798 \fi}}
8799 \def\bbl@hebreleapsedmonths#1#2{%
8800 {\countdef\tmpa=0
8801 \countdef\tmpb=1
8802 \countdef\tmpc=2
8803 \tmpa=#1\relax
8804 \advance \tmpa by -1
8805 #2=\tmpa
8806 \divide #2 by 19
8807 \multiply #2 by 235
8808 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8809 \tmpc=\tmpb
8810 \multiply \tmpb by 12
8811 \advance #2 by \tmpb
8812 \multiply \tmpc by 7
8813 \advance \tmpc by 1
8814 \divide \tmpc by 19
8815 \advance #2 by \tmpc
8816 \global\bbl@cntcommon=#2}%
8817 #2=\bbl@cntcommon}
8818 \def\bbl@hebreleapseddays#1#2{%
8819 {\countdef\tmpa=0
8820 \countdef\tmpb=1
8821 \countdef\tmpc=2
8822 \bbl@hebreleapsedmonths{#1}{#2}%
8823 \tmpa=#2\relax
8824 \multiply \tmpa by 13753
8825 \advance \tmpa by 5604
8826 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8827 \divide \tmpa by 25920
8828 \multiply #2 by 29
8829 \advance #2 by 1
8830 \advance #2 by \tmpa
8831 \bbl@remainder{#2}{7}{\tmpa}%
8832 \ifnum \tmpc < 19440
8833 \ifnum \tmpc < 9924

```

```

8834     \else
8835         \ifnum \tmpa=2
8836             \bbl@checkleaphebyear{#1}% of a common year
8837             \ifbbl@hebrleap
8838                 \else
8839                     \advance #2 by 1
8840                 \fi
8841             \fi
8842         \fi
8843         \ifnum \tmpc < 16789
8844             \else
8845                 \ifnum \tmpa=1
8846                     \advance #1 by -1
8847                     \bbl@checkleaphebyear{#1}% at the end of leap year
8848                     \ifbbl@hebrleap
8849                         \advance #2 by 1
8850                     \fi
8851                 \fi
8852             \fi
8853         \else
8854             \advance #2 by 1
8855         \fi
8856         \bbl@remainder{#2}{7}{\tmpa}%
8857         \ifnum \tmpa=0
8858             \advance #2 by 1
8859         \else
8860             \ifnum \tmpa=3
8861                 \advance #2 by 1
8862             \else
8863                 \ifnum \tmpa=5
8864                     \advance #2 by 1
8865                 \fi
8866             \fi
8867         \fi
8868         \global\bbl@cntcommon=#2\relax}%
8869     #2=\bbl@cntcommon}
8870 \def\bbl@daysinhebyear#1#2{%
8871     {\countdef\tmpe=12
8872     \bbl@hebreleaseddays{#1}{\tmpe}%
8873     \advance #1 by 1
8874     \bbl@hebreleaseddays{#1}{#2}%
8875     \advance #2 by -\tmpe
8876     \global\bbl@cntcommon=#2}%
8877     #2=\bbl@cntcommon}
8878 \def\bbl@hebrdayspriormonths#1#2#3{%
8879     {\countdef\tmpf= 14
8880     #3=\ifcase #1
8881         0 \or
8882         0 \or
8883         30 \or
8884         59 \or
8885         89 \or
8886         118 \or
8887         148 \or
8888         148 \or
8889         177 \or
8890         207 \or
8891         236 \or
8892         266 \or
8893         295 \or
8894         325 \or
8895         400
8896     \fi

```

```

8897 \bbl@checkleaphebryear{#2}%
8898 \ifbbl@hebrleap
8899 \ifnum #1 > 6
8900 \advance #3 by 30
8901 \fi
8902 \fi
8903 \bbl@daysinhebryear{#2}{\tmpf}%
8904 \ifnum #1 > 3
8905 \ifnum \tmpf=353
8906 \advance #3 by -1
8907 \fi
8908 \ifnum \tmpf=383
8909 \advance #3 by -1
8910 \fi
8911 \fi
8912 \ifnum #1 > 2
8913 \ifnum \tmpf=355
8914 \advance #3 by 1
8915 \fi
8916 \ifnum \tmpf=385
8917 \advance #3 by 1
8918 \fi
8919 \fi
8920 \global\bbl@cntcommon=#3\relax}%
8921 #3=\bbl@cntcommon}
8922 \def\bbl@absfromhebr#1#2#3#4{%
8923 {#4=#1\relax
8924 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8925 \advance #4 by #1\relax
8926 \bbl@hebrrelapseddays{#3}{#1}%
8927 \advance #4 by #1\relax
8928 \advance #4 by -1373429
8929 \global\bbl@cntcommon=#4\relax}%
8930 #4=\bbl@cntcommon}
8931 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8932 {\countdef\tmpx= 17
8933 \countdef\tmpy= 18
8934 \countdef\tmpz= 19
8935 #6=#3\relax
8936 \global\advance #6 by 3761
8937 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8938 \tmpz=1 \tmpy=1
8939 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8940 \ifnum \tmpx > #4\relax
8941 \global\advance #6 by -1
8942 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8943 \fi
8944 \advance #4 by -\tmpx
8945 \advance #4 by 1
8946 #5=#4\relax
8947 \divide #5 by 30
8948 \loop
8949 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8950 \ifnum \tmpx < #4\relax
8951 \advance #5 by 1
8952 \tmpy=\tmpx
8953 \repeat
8954 \global\advance #5 by -1
8955 \global\advance #4 by -\tmpy}}
8956 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8957 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8958 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8959 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax

```

```

8960 \bbl@hebrfromgreg
8961 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8962 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8963 \edef#4{\the\bbl@hebyear}%
8964 \edef#5{\the\bbl@hebrmonth}%
8965 \edef#6{\the\bbl@hebrday}%
8966 </ca-hebrew>

```

13.3. Persian

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

```

8967 <*ca-persian>
8968 <@Compute Julian day@>
8969 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8970 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8971 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8972 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8973 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8974 \bbl@afterfi\expandafter\@gobble
8975 \fi\fi
8976 {\bbl@error{year-out-range}{2013-2050}{}}}%
8977 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8978 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8979 \edef\bbl@tempc{\fpeval{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8980 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8981 \ifnum\bbl@tempc<\bbl@tempb
8982 \edef\bbl@tempa{\fpeval{\bbl@tempa-1}}% go back 1 year and redo
8983 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8984 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8985 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8986 \fi
8987 \edef#4{\fpeval{\bbl@tempa-621}}% set Jalali year
8988 \edef#6{\fpeval{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8989 \edef#5{\fpeval{% set Jalali month
8990 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8991 \edef#6{\fpeval{% set Jalali day
8992 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}%
8993 </ca-persian>

```

13.4. Coptic and Ethiopic

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

```

8994 <*ca-coptic>
8995 <@Compute Julian day@>
8996 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8997 \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8998 \edef\bbl@tempc{\fpeval{\bbl@tempd - 1825029.5}}%
8999 \edef#4{\fpeval{%
9000 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
9001 \edef\bbl@tempc{\fpeval{%
9002 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
9003 \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
9004 \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
9005 </ca-coptic>
9006 <*ca-ethiopic>
9007 <@Compute Julian day@>
9008 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%

```

```

9009 \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
9010 \edef\bbl@tempc{\fpeval{\bbl@tempd - 1724220.5}}%
9011 \edef#4{\fpeval{%
9012   floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
9013 \edef\bbl@tempc{\fpeval{%
9014   \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
9015 \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
9016 \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
9017 </ca-ethiopic>

```

13.5. Julian

Based on [ReinDersh].

```

9018 <*ca-julian>
9019 <@Compute Julian day@>
9020 \def\bbl@ca@julian#1-#2-#3\@@#4#5#6{%
9021   \edef\bbl@tempj{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + .5}}%
9022   \edef\bbl@tempa{\fpeval{\bbl@tempj + 32082.5}}%
9023   \edef\bbl@tempb{\fpeval{floor((4 * \bbl@tempa + 3) / 1461)}}%
9024   \edef\bbl@tempc{\fpeval{\bbl@tempa - floor(1461*\bbl@tempb/4)}}%
9025   \edef\bbl@tempd{\fpeval{floor((5 * \bbl@tempc + 2) / 153)}}%
9026   \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
9027   \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
9028   \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}%
9029 </ca-julian>

```

The Indian National Calendar, Saka era.

```

9030 <*ca-indian>
9031 <@Compute Julian day@>
9032 \def\bbl@sakachaitra#1{%
9033   \bbl@cs@jd{#1}{3}%
9034   \fpeval{ ((\bbl@fpmmod{#1}{4})==0 && \bbl@fpmmod{#1}{100}!=0)
9035   || \bbl@fpmmod{#1}{400}==0) ? 21 : 22 }}%
9036 \def\bbl@ca@indian#1-#2-#3\@@#4#5#6{%
9037   \edef\bbl@tempa{\fpeval{\bbl@cs@jd{#1}{#2}{#3}}}%
9038   \edef\bbl@tempb{\fpeval{\bbl@sakachaitra{#1}}}%
9039   \edef\bbl@tempc{\fpeval{\bbl@tempa < \bbl@tempb ? #1-1 : #1}}%
9040   \edef\bbl@tempd{\fpeval{\bbl@tempa-\bbl@sakachaitra{\bbl@tempc}}}%
9041   \edef\bbl@tempe{\fpeval{ ((\bbl@fpmmod{\bbl@tempc}{4})==0 &&
9042     \bbl@fpmmod{\bbl@tempc}{100}!=0) || \bbl@fpmmod{\bbl@tempc}{400}==0) ? 31 : 30 }}%
9043   \edef#4{\fpeval{\bbl@tempc-78}}%
9044   \edef#5{\fpeval{\bbl@tempd<\bbl@tempe ? 1 :
9045     (\bbl@tempd-\bbl@tempe < 155 ?
9046       2+trunc((\bbl@tempd-\bbl@tempe)/31,0) :
9047       7+trunc((\bbl@tempd-\bbl@tempe-155)/30,0))}}%
9048   \edef#6{\fpeval{\bbl@tempd<\bbl@tempe ? \bbl@tempd+1 :
9049     (\bbl@tempd-\bbl@tempe < 155 ?
9050       \bbl@fpmmod{\bbl@tempd-\bbl@tempe}{31}+1 :
9051       \bbl@fpmmod{\bbl@tempd-\bbl@tempe-155}{30}+1)}}%
9052 </ca-indian>
9053 %

```

13.6. Buddhist

That's very simple.

```

9054 <*ca-buddhist>
9055 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
9056   \edef#4{\number\numexpr#1+543\relax}%
9057   \edef#5{#2}%
9058   \edef#6{#3}%
9059 </ca-buddhist>
9060 %

```

```

9061 % \subsection{Chinese}
9062 %
9063 % Brute force, with the Julian day of first day of each month. The
9064 % table has been computed with the help of \textsf{python-lunardate} by
9065 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
9066 % is 2015-2044.
9067 %
9068 % \begin{macrocode}
9069 <*ca-chinese>
9070 \ExplSyntaxOn
9071 <@Compute Julian day@>
9072 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
9073 \edef\bbl@tempd{\fpeval{%
9074 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
9075 \count@ \z@
9076 \@tempcnta=2015
9077 \bbl@foreach\bbl@cs@chinese@data{%
9078 \ifnum##1>\bbl@tempd\else
9079 \advance\count@\@ne
9080 \ifnum\count@>12
9081 \count@\@ne
9082 \advance\@tempcnta\@ne\fi
9083 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
9084 \ifin@
9085 \advance\count@\m@ne
9086 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
9087 \else
9088 \edef\bbl@tempe{\the\count@}%
9089 \fi
9090 \edef\bbl@tempb{##1}%
9091 \fi}%
9092 \edef#4{\the\@tempcnta}%
9093 \edef#5{\bbl@tempe}%
9094 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
9095 \def\bbl@cs@chinese@leap{%
9096 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
9097 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
9098 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
9099 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
9100 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
9101 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
9102 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
9103 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
9104 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
9105 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
9106 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
9107 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
9108 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
9109 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
9110 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
9111 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
9112 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
9113 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
9114 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
9115 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
9116 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
9117 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
9118 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
9119 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
9120 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
9121 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
9122 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
9123 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%

```

```

9124 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
9125 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
9126 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
9127 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
9128 10896,10926,10956,10986,11015,11045,11074,11103}
9129 \ExplSyntaxOff
9130 </ca-chinese>

```

14. Support for Plain T_EX (plain.def)

14.1. Not renaming hyphen.tex

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based T_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 `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the `babel` package. If you load each of them with `iniTEX`, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

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

```

9131 <*\bplain | blplain>
9132 \catcode`\{=1 % left brace is begin-group character
9133 \catcode`\}=2 % right brace is end-group character
9134 \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).

```

9135 \openin 0 hyphen.cfg
9136 \ifeof0
9137 \else
9138   \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.

```

9139 \def\input #1 {%
9140   \let\input\a
9141   \a hyphen.cfg
9142   \let\a\undefined
9143 }
9144 \fi
9145 </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`.

```

9146 <bplain>\a plain.tex
9147 <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.

```

9148 <bplain>\def\fmtname{babel-plain}
9149 <blplain>\def\fmtname{babel-lplain}

```

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

14.2. Emulating some $\text{\LaTeX}$ 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).

```
9150 <<{*Emulate LaTeX}>> ≡
9151 \def\@empty{}
9152 \def\loadlocalcfg#1{%
9153   \openin0#1.cfg
9154   \ifeof0
9155     \closein0
9156   \else
9157     \closein0
9158     {\immediate\write16{*****}%
9159      \immediate\write16{* Local config file #1.cfg used}%
9160      \immediate\write16{*}%
9161     }
9162     \input #1.cfg\relax
9163   \fi
9164   \@endofldf}
```

14.3. General tools

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

```
9165 \long\def\@firstofone#1{#1}
9166 \long\def\@firstoftwo#1#2{#1}
9167 \long\def\@secondoftwo#1#2{#2}
9168 \def\@nnil{\@nil}
9169 \def\@gobbletwo#1#2{}
9170 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
9171 \def\@staror@long#1{%
9172   \@ifstar
9173   {\let\l@ngrel@x\relax#1}%
9174   {\let\l@ngrel@x\long#1}}
9175 \let\l@ngrel@x\relax
9176 \def\@car#1#2\@nil{#1}
9177 \def\@cdr#1#2\@nil{#2}
9178 \let\@typeset@protect\relax
9179 \let\protected@edef\edef
9180 \long\def\@gobble#1{}
9181 \edef\@backslashchar{\expandafter\@gobble\string\}
9182 \def\strip@prefix#1>{}
9183 \def\g@addto@macro#1#2{{%
9184   \toks@\expandafter{#1#2}%
9185   \xdef#1{\the\toks@}}}
9186 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
9187 \def\@nameuse#1{\csname #1\endcsname}
9188 \def\@ifundefined#1{%
9189   \expandafter\ifx\csname#1\endcsname\relax
9190     \expandafter\@firstoftwo
9191   \else
9192     \expandafter\@secondoftwo
9193   \fi}
9194 \def\@expandtwoargs#1#2#3{%
9195   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9196 \def\zap@space#1 #2{%
9197   #1%
9198   \ifx#2\@empty\else\expandafter\zap@space\fi
9199   #2}
9200 \let\bbl@trace\@gobble
9201 \def\bbl@error#1{% Implicit #2#3#4}
```

```

9202 \begingroup
9203   \catcode`\=0   \catcode`\==12 \catcode`\`=12
9204   \catcode`\^M=5 \catcode`\%=14
9205   \input errbabel.def
9206 \endgroup
9207 \bbl@error{#1}}
9208 \def\bbl@warning#1{%
9209   \begingroup
9210     \newlinechar=`^^J
9211     \def\{`^^J(babel) }%
9212     \message{\{#1}%
9213   \endgroup}
9214 \let\bbl@infowarn\bbl@warning
9215 \def\bbl@info#1{%
9216   \begingroup
9217     \newlinechar=`^^J
9218     \def\{`^^J}%
9219     \wlog{#1}%
9220   \endgroup}

```

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

```

9221 \ifx\@preamblecmds\undefined
9222   \def\@preamblecmds{}
9223 \fi
9224 \def\@onlypreamble#1{%
9225   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9226     \@preamblecmds\do#1}}
9227 \@onlypreamble\@onlypreamble

```

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

```

9228 \def\begindocument{%
9229   \@begindocumenthook
9230   \global\let\@begindocumenthook\undefined
9231   \def\do##1{\global\let##1\undefined}%
9232   \@preamblecmds
9233   \global\let\do\noexpand}
9234 \ifx\@begindocumenthook\undefined
9235   \def\@begindocumenthook{}
9236 \fi
9237 \@onlypreamble\@begindocumenthook
9238 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

9239 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
9240 \@onlypreamble\AtEndOfPackage
9241 \def\@endoflfd{}
9242 \@onlypreamble\@endoflfd
9243 \let\bbl@afterlang\@empty
9244 \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.

```

9245 \catcode`\&=\z@
9246 \ifx&\if@files\undefined
9247   \expandafter\let\csname if@files\expandafter\endcsname
9248     \csname iffalse\endcsname
9249 \fi
9250 \catcode`\&=4

```

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

```

9251 \def\newcommand{\@star@or@long\new@command}
9252 \def\new@command#1{%
9253   \@testopt{\@newcommand#1}0}
9254 \def\@newcommand#1[#2]{%
9255   \@ifnextchar [{\@xargdef#1[#2]}%
9256     {\@argdef#1[#2]}}
9257 \long\def\@argdef#1[#2]#3{%
9258   \@yargdef#1\@ne{#2}{#3}}
9259 \long\def\@xargdef#1[#2][#3]#4{%
9260   \expandafter\def\expandafter#1\expandafter{%
9261     \expandafter\@protected@testopt\expandafter #1%
9262     \csname\string#1\expandafter\endcsname{#3}}}%
9263   \expandafter\@yargdef \csname\string#1\endcsname
9264   \tw@{#2}{#4}}
9265 \long\def\@yargdef#1#2#3{%
9266   \@tempcnta#3\relax
9267   \advance \@tempcnta \@ne
9268   \let\@hash@\relax
9269   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9270   \@tempcntb #2%
9271   \@whilenum \@tempcntb < \@tempcnta
9272   \do{%
9273     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
9274     \advance \@tempcntb \@ne}%
9275   \let\@hash@###%
9276   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9277 \def\providecommand{\@star@or@long\provide@command}
9278 \def\provide@command#1{%
9279   \begingroup
9280     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9281   \endgroup
9282   \expandafter\ifundefined\@gtempa
9283     {\def\reserved@a{\new@command#1}}%
9284     {\let\reserved@a\relax
9285     \def\reserved@a{\new@command\reserved@a}}%
9286   \reserved@a}%

9287 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9288 \def\declare@robustcommand#1{%
9289   \edef\reserved@a{\string#1}%
9290   \def\reserved@b{#1}%
9291   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9292   \edef#1{%
9293     \ifx\reserved@a\reserved@b
9294       \noexpand\x@protect
9295       \noexpand#1%
9296     \fi
9297     \noexpand\protect
9298     \expandafter\noexpand\csname
9299       \expandafter\@gobble\string#1 \endcsname
9300   }%
9301   \expandafter\new@command\csname
9302     \expandafter\@gobble\string#1 \endcsname
9303 }
9304 \def\x@protect#1{%
9305   \ifx\protect\@typeset@protect\else
9306     \x@protect#1%
9307   \fi
9308 }
9309 \catcode`\&=\z@ % Trick to hide conditionals
9310 \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`.

```

9311 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9312 \catcode`\&=4
9313 \ifx\in@\@undefined
9314 \def\in@#1#2{%
9315 \def\in@@##1##2##3\in@@{%
9316 \ifx\in@@##2\in@false\else\in@true\fi}%
9317 \in@@##2#1\in@\in@}
9318 \else
9319 \let\bbl@tempa\@empty
9320 \fi
9321 \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).

```

9322 \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.

```

9323 \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.

```

9324 \ifx\@tempcnta\@undefined
9325 \csname newcount\endcsname\@tempcnta\relax
9326 \fi
9327 \ifx\@tempcntb\@undefined
9328 \csname newcount\endcsname\@tempcntb\relax
9329 \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`).

```

9330 \ifx\bye\@undefined
9331 \advance\count10 by -2\relax
9332 \fi
9333 \ifx\@ifnextchar\@undefined
9334 \def\@ifnextchar#1#2#3{%
9335 \let\reserved@d=#1%
9336 \def\reserved@a{#2}\def\reserved@b{#3}%
9337 \futurelet\@let@token\@ifnch}
9338 \def\@ifnch{%
9339 \ifx\@let@token\@sptoken
9340 \let\reserved@c\@xifnch
9341 \else
9342 \ifx\@let@token\reserved@d
9343 \let\reserved@c\reserved@a
9344 \else
9345 \let\reserved@c\reserved@b
9346 \fi
9347 \fi
9348 \reserved@c}
9349 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
9350 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
9351 \fi
9352 \def\@testopt#1#2{%
9353 \@ifnextchar[#{1}{#1[#{2}]}}
9354 \def\@protected@testopt#1{%
9355 \ifx\protect\@typeset@protect
9356 \expandafter\@testopt

```

```

9357 \else
9358 \@@protect#1%
9359 \fi}
9360 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9361 #2\relax}\fi}
9362 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9363 \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9364 \def\DeclareTextCommand{%
9365 \@@dec@text@cmd\providecommand
9366 }
9367 \def\ProvideTextCommand{%
9368 \@@dec@text@cmd\providecommand
9369 }
9370 \def\DeclareTextSymbol#1#2#3{%
9371 \@@dec@text@cmd\chardef#1{#2}#3\relax
9372 }
9373 \def\@dec@text@cmd#1#2#3{%
9374 \expandafter\def\expandafter#2%
9375 \expandafter{%
9376 \csname#3-cmd\expandafter\endcsname
9377 \expandafter#2%
9378 \csname#3\string#2\endcsname
9379 }%
9380 % \let\@ifdefinable\@rc@ifdefinable
9381 \expandafter#1\csname#3\string#2\endcsname
9382 }
9383 \def\@current@cmd#1{%
9384 \ifx\protect\@typeset@protect\else
9385 \noexpand#1\expandafter\@gobble
9386 \fi
9387 }
9388 \def\@changed@cmd#1#2{%
9389 \ifx\protect\@typeset@protect
9390 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9391 \expandafter\ifx\csname ?\string#1\endcsname\relax
9392 \expandafter\def\csname ?\string#1\endcsname{%
9393 \@changed@x@err{#1}%
9394 }%
9395 \fi
9396 \global\expandafter\let
9397 \csname\cf@encoding\string#1\expandafter\endcsname
9398 \csname ?\string#1\endcsname
9399 \fi
9400 \csname\cf@encoding\string#1%
9401 \expandafter\endcsname
9402 \else
9403 \noexpand#1%
9404 \fi
9405 }
9406 \def\@changed@x@err#1{%
9407 \errhelp{Your command will be ignored, type <return> to proceed}%
9408 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9409 \def\DeclareTextCommandDefault#1{%
9410 \DeclareTextCommand#1?%
9411 }
9412 \def\ProvideTextCommandDefault#1{%
9413 \ProvideTextCommand#1?%
9414 }
9415 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd

```

```

9416 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9417 \def\DeclareTextAccent#1#2#3{%
9418   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9419 }
9420 \def\DeclareTextCompositeCommand#1#2#3#4{%
9421   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9422   \edef\reserved@b{\string##1}%
9423   \edef\reserved@c{%
9424     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9425   \ifx\reserved@b\reserved@c
9426     \expandafter\expandafter\expandafter\ifx
9427       \expandafter\@car\reserved@a\relax\relax\@nil
9428       \@text@composite
9429     \else
9430       \edef\reserved@b##1{%
9431         \def\expandafter\noexpand
9432           \csname#2\string#1\endcsname###1{%
9433             \noexpand\@text@composite
9434             \expandafter\noexpand\csname#2\string#1\endcsname
9435             ###1\noexpand\@empty\noexpand\@text@composite
9436             {##1}%
9437           }%
9438         }%
9439       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9440     \fi
9441     \expandafter\def\csname\expandafter\string\csname
9442       #2\endcsname\string#1-\string#3\endcsname{#4}
9443   \else
9444     \errhelp{Your command will be ignored, type <return> to proceed}%
9445     \errmessage{\string\DeclareTextCompositeCommand\space used on
9446       inappropriate command \protect#1}
9447   \fi
9448 }
9449 \def\@text@composite#1#2#3\@text@composite{%
9450   \expandafter\@text@composite@x
9451   \csname\string#1-\string#2\endcsname
9452 }
9453 \def\@text@composite@x#1#2{%
9454   \ifx#1\relax
9455     #2%
9456   \else
9457     #1%
9458   \fi
9459 }
9460 %
9461 \def\@strip@args#1:#2-#3\@strip@args{#2}
9462 \def\DeclareTextComposite#1#2#3#4{%
9463   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9464   \bgroup
9465     \lccode`\@=#4%
9466     \lowercase{%
9467       \egroup
9468       \reserved@a @%
9469     }%
9470 }
9471 %
9472 \def\UseTextSymbol#1#2{#2}
9473 \def\UseTextAccent#1#2#3{}
9474 \def\@use@text@encoding#1{}
9475 \def\DeclareTextSymbolDefault#1#2{%
9476   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9477 }
9478 \def\DeclareTextAccentDefault#1#2{%

```

```

9479 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9480 }
9481 \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.

```

9482 \DeclareTextAccent{"}{OT1}{127}
9483 \DeclareTextAccent{'}{OT1}{19}
9484 \DeclareTextAccent{^}{OT1}{94}
9485 \DeclareTextAccent{\`}{OT1}{18}
9486 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9487 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9488 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
9489 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9490 \DeclareTextSymbol{\textquoteright}{OT1}{''}
9491 \DeclareTextSymbol{\i}{OT1}{16}
9492 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9493 \ifx\scriptsize\undefined
9494 \let\scriptsize\sevenrm
9495 \fi

```

And a few more “dummy” definitions.

```

9496 \def\language{english}%
9497 \let\bbl@opt@shorthands@nnil
9498 \def\bbl@ifshorthand#1#2#3{#2}%
9499 \let\bbl@language@opts@empty
9500 \let\bbl@provide@locale\relax
9501 \ifx\babeloptionstrings\undefined
9502 \let\bbl@opt@strings@nnil
9503 \else
9504 \let\bbl@opt@strings\babeloptionstrings
9505 \fi
9506 \def\BabelStringsDefault{generic}
9507 \def\bbl@tempa{normal}
9508 \ifx\babeloptionmath\bbl@tempa
9509 \def\bbl@mathnormal{\noexpand\textormath}
9510 \fi
9511 \def\AfterBabelLanguage#1#2{}
9512 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9513 \let\bbl@afterlang\relax
9514 \def\bbl@opt@safe{BR}
9515 \ifx@uclclist\undefined\let@uclclist@empty\fi
9516 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9517 \expandafter\newif\csname ifbbl@single\endcsname
9518 \chardef\bbl@bidimode\z@
9519 <</Emulate LaTeX>>

```

A proxy file:

```

9520 <*\plain>
9521 \input babel.def
9522 </\plain>

```

15. Acknowledgements

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

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

Barbara Beeton has helped in improving the manual.

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

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national $\LaTeX$ 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, *$\LaTeX$, 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 $\LaTeX$ is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using $\LaTeX$* , Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).