

Babel

Code

Version 26.10

2026/08/08

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	67
4.21	French spacing (again)	73
4.22	Handle language system	75
4.23	Numerals	75
4.24	Casing	77
4.25	Getting info	77
4.26	BCP 47 related commands	78
5	Adjusting the Babel behavior	79
5.1	Cross referencing macros	82
5.2	Layout	84
5.3	Marks	86
5.4	Other packages	87
5.4.1	ifthen	87
5.4.2	varioref	87
5.4.3	hhline	88
5.5	Encoding and fonts	88
5.6	Basic bidi support	90
5.7	Local Language Configuration	93
5.8	Language options	94

6	The kernel of Babel	98
7	Error messages	98
8	Loading hyphenation patterns	102
9	luatex + xetex: common stuff	106
10	Hooks for XeTeX and LuaTeX	109
10.1	XeTeX	109
10.2	Support for interchar	111
10.3	Layout	113
10.4	8-bit TeX	115
10.5	LuaTeX	115
10.6	Southeast Asian scripts	122
10.7	CJK line breaking	123
10.8	Arabic justification	125
10.9	Common stuff	130
10.10	Automatic fonts and ids switching	130
10.11	Bidi	137
10.12	Layout	139
10.13	Lua: transforms	148
10.14	Lua: Auto bidi with basic and basic-r	158
11	Data for CJK	170
12	The ‘nil’ language	170
13	Calendars	171
13.1	Islamic	171
13.2	Hebrew	173
13.3	Persian	177
13.4	Coptic and Ethiopic	178
13.5	Julian	178
13.6	Buddhist	178
14	Support for Plain T_EX (plain.def)	180
14.1	Not renaming hyphen.tex	180
14.2	Emulating some L ^A T _E X features	181
14.3	General tools	181
14.4	Encoding related macros	185
15	Acknowledgements	187

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=26.10>>
2 <<date=2026/08/08>>
```

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

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

```

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

```

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

```

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

```

\bbl@afterelse

\bbl@afterfi Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an \if-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 `\ifcsname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

```

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

```

\bbl@ifblank A tool from url, by Donald Arseneau, which tests if a string is empty or space. The companion macros tests if a macro is defined with some ‘real’ value, 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\bbl@cntchars{##1}>4
531 \lowercase{\def\bbl@tempv{##1}}%
532 \fi\fi\fi\fi}%
533 \bbl@exp{\bbl@bcpllookup{i\bbl@templ}\bbl@temps}\bbl@tempr}\bbl@tempv}}
534 \def\bbl@bcpllookup@try#1{%
535 \ifx\bbl@bcpl\relax
536 \IfFileExists{babel-#1\bbl@tempe.ini}{\edef\bbl@bcpl{#1\bbl@tempe}}{}%
537 \fi}
538 \def\bbl@bcpllookup@i#1#2#3#4{%
539 \let\bbl@bcpl\relax
540 \def\bbl@tempe{#4}%
541 \ifx\bbl@tempe\@empty\else
542 \edef\bbl@tempe{-#4}%
543 \fi
544 \edef\bbl@tempa{#2#3\bbl@tempe}% en
545 \ifx\bbl@tempa\@empty
546 \bbl@bcpllookup@try{#1}%
547 \else
548 \edef\bbl@tempa{#3\bbl@tempe}% en-Latn
549 \ifx\bbl@tempa\@empty
550 \bbl@bcpllookup@try{#1-#2}%
551 \bbl@bcpllookup@try{#1}%
552 \else
553 \edef\bbl@tempa{#2\bbl@tempe}% en-US, en-001
554 \ifx\bbl@tempa\@empty
555 \bbl@bcpllookup@try{#1-#3}%
556 \bbl@bcpllookup@try{#1}%
557 \else % en-Latn-US
558 \bbl@bcpllookup@try{#1-#2-#3}%
559 \bbl@bcpllookup@try{#1-#2}%
560 \bbl@bcpllookup@try{#1-#3}%
561 \bbl@bcpllookup@try{#1}%
562 \fi
563 \fi
564 \fi
565 \ifx\bbl@bcpl\relax
566 \ifx\bbl@tempe\@empty\else\bbl@bcpllookup@i{#1}{#2}{#3}}\fi%
567 \fi}
568 %
569 \let\bbl@initoload\relax

```

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

```

570 \def\iflanguage#1{%
571 \bbl@iflanguage{#1}{%
572 \ifnum\cscname{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\bbl@select@type\z@
578 \edef\selectlanguage{%
579 \noexpand\protect
580 \expandafter\noexpand\cscname 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 $\text{\TeX 2.}\epsilon$. 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 $\text{\LaTeX 2}_{\epsilon}$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

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

```

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,##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}}
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 *ltxex*.

```

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@filesw
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\bbled@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\bbled@tempa\@firstoftwo
1277 \if\string^#2%
1278 \def\bbled@tempa{\noexpand\textormath}%
1279 \else
1280 \ifx\bbled@mathnormal\undefined\else
1281 \let\bbled@tempa\bbled@mathnormal
1282 \fi
1283 \fi
1284 \expandafter\edef\csname active@char#2\endcsname{%
1285 \bbled@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 bbled@doactive#2\endcsname
1292 \noexpand\fi}%
1293 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1294 \bbled@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 \bbled@csarg\edef{active@#2}{%
1297 \noexpand\active@prefix\noexpand#1%
1298 \expandafter\noexpand\csname active@char#2\endcsname}%
1299 \bbled@csarg\edef{normal@#2}{%
1300 \noexpand\active@prefix\noexpand#1%
1301 \expandafter\noexpand\csname normal@char#2\endcsname}%
1302 \bbled@ncarg\let#1\bbled@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 \bbled@active@def#2\user@group{user@active}{language@active}%
1304 \bbled@active@def#2\language@group{language@active}{system@active}%
1305 \bbled@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@activesfalse

```

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

```

1364 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

\bbl@activate

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

```

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     \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,}{,#1,}%
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 \babe\nullhyphen
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 discretionary 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 $\TeX$'s `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```
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,alpha,%
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\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2280 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw}%
2281 % == init ==
2282 \ifx\bbl@screset\@undefined
2283 \bbl@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\bbl@KVP@import\@nnil
2290 \let\bbl@KVP@import\bbl@KVP@@import
2291 \fi
2292 % == date (as option) ==
2293 % \ifx\bbl@KVP@date\@nnil\else
2294 % \fi
2295 % ==
2296 \let\bbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2297 \ifcase\bbl@howloaded
2298 \let\bbl@lbfkflag\@empty % new
2299 \else
2300 \ifx\bbl@KVP@hyphenrules\@nnil\else
2301 \let\bbl@lbfkflag\@empty
2302 \fi
2303 \ifx\bbl@KVP@import\@nnil\else
2304 \let\bbl@lbfkflag\@empty
2305 \fi
2306 \fi
2307 % == import, captions ==
2308 \ifx\bbl@KVP@import\@nnil\else
2309 \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
2310 {\ifx\bbl@initoload\relax
2311 \begingroup
2312 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2313 \bbl@input@texini{##2}%
2314 \endgroup
2315 \else
2316 \xdef\bbl@KVP@import{\bbl@initoload}%
2317 \fi}%
2318 {}%
2319 \let\bbl@KVP@date\@empty
2320 \fi
2321 \let\bbl@KVP@captions@@\bbl@KVP@captions
2322 \ifx\bbl@KVP@captions\@nnil
2323 \let\bbl@KVP@captions\bbl@KVP@import
2324 \fi
2325 % ==
2326 \ifx\bbl@KVP@transforms\@nnil\else
2327 \bbl@replace\bbl@KVP@transforms{ }{,}%
2328 \fi
2329 % ==
2330 \ifx\bbl@KVP@mapdot\@nnil\else
2331 \def\bbl@tempa{\@empty}%
2332 \ifx\bbl@KVP@mapdot\bbl@tempa\else
2333 \bbl@exp{\gdef<\bbl@map@@.\@{\language\name}>{\bbl@KVP@mapdot}}}%
2334 \fi
2335 \fi
2336 % Load ini
2337 % -----
2338 \ifcase\bbl@howloaded
2339 \bbl@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\language\@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@\language}%
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\{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{\bbl@tempa##3 \@}%
2790          \let\localename\language
2791          \bbl@id@assign
2792          \def\bbl@elt####1####2####3{}}%
2793         {}}%
2794       \bbl@inidata
2795     \fi
2796     \bbl@csarg\xdef{lini@\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@exp{\\ \bbl@add@list\\ \bbl@ini@loaded{\language}}%
2803     \bbl@tglobal\bbl@ini@loaded
2804     \@ifpackagewith{babel}{licr=unextended}{}%

```

```

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

```

2828 \def\bbl@extend@ini@aux#1{%
2829   \bbl@startcommands*{#1}{captions}%
2830   % Activate captions/... and modify exports
2831   \bbl@csarg\def{inikv@captions.licr}##1##2{%
2832     \setlocalecaption{#1}{##1}{##2}}%
2833   \def\bbl@inikv@captions##1##2{%
2834     \bbl@ini@captions@aux{##1}{##2}}%
2835   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2836   \def\bbl@exportkey##1##2##3{%
2837     \bbl@ifunset{bbl@kv@##2}{}%
2838     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2839       \bbl@exp{\global\let<bbl@##1\language>\<bbl@kv@##2>}}%
2840     \fi}%
2841   % As with \bbl@read@ini, but with some changes
2842   \bbl@read@ini@aux
2843   \bbl@ini@exports\tw@
2844   % Update inidata@lang by pretending the ini is read.
2845   \def\bbl@elt##1##2##3{%
2846     \def\bbl@section{##1}%
2847     \bbl@iniline##2=##3\bbl@iniline}%
2848     \csname bbl@inidata@#1\endcsname
2849     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2850   \StartBabelCommands*{#1}{date}% And from the import stuff
2851   \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2852   \bbl@savetoday
2853   \bbl@savestate
2854   \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2855 \def\bbl@ini@calendar#1{%
2856   \lowercase{\def\bbl@tempa{=##1=}}%
2857   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2858   \bbl@replace\bbl@tempa{=date.}{}%
2859   \in@{.licr=}{#1}%
2860   \ifin@
2861     \ifcase\bbl@engine
2862       \bbl@replace\bbl@tempa{.licr=}{}%
2863     \else

```

```

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

```

2876 \def\bbl@renewinikey#1/#2\@#3{%
2877 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2878 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2879 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2880 \bbl@trim\toks@{#3}% value
2881 \bbl@exp{%
2882 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2883 \\g@addto@macro\\bbl@inidata{%
2884 \\\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.

```

2885 \def\bbl@exportkey#1#2#3{%
2886 \bbl@ifunset{\bbl@kv@#2}%
2887 {\bbl@csarg\gdef{#1@\language}\{#3}}%
2888 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
2889 \bbl@csarg\gdef{#1@\language}\{#3}%
2890 \else
2891 \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
2892 \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.

```

2893 \def\bbl@iniwarning#1{%
2894 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2895 {\bbl@warning{%
2896 From babel-\bbl@cs{lini@\language}.ini:\\%
2897 \bbl@cs{@kv@identification.warning#1}\\%
2898 Reported}}}
2899 %
2900 \let\bbl@release@transforms\@empty
2901 \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).

```

2902 \def\bbl@ini@exports#1{%
2903   % Identification always exported
2904   \bbl@iniwarning{}%
2905   \ifcase\bbl@engine
2906     \bbl@iniwarning{.pdflatex}%
2907   \or
2908     \bbl@iniwarning{.lualatex}%
2909   \or
2910     \bbl@iniwarning{.xelatex}%
2911   \fi%
2912   \bbl@exportkey{lllevel}{identification.load.level}{}%
2913   \bbl@exportkey{elname}{identification.name.english}{}%
2914   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2915     {\csname bbl@elname@\language\endcsname}}%
2916   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2917   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2918   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2919   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2920   \bbl@exportkey{esname}{identification.script.name}{}%
2921   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2922     {\csname bbl@esname@\language\endcsname}}%
2923   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2924   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2925   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2926   \bbl@exportkey{vbc}{identification.variant.tag.bcp47}{}%
2927   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2928   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2929   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2930   % Also maps bcp47 -> language
2931   \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
2932   \ifcase\bbl@engine\or
2933     \directlua{%
2934       Babel.locale_props[\the\bbl@cs{id@\language}].script
2935       = '\bbl@cl{sbc}}'%
2936   \fi
2937   % Conditional
2938   \ifnum#1>\z@      % -1 or 0 = only info, 1 = basic, 2 = (re)new
2939     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2940     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2941     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2942     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2943     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2944     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2945     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2946     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2947     \bbl@exportkey{intsp}{typography.intraspaces}{}%
2948     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2949     \bbl@exportkey{chrng}{characters.ranges}{}%
2950     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2951     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2952     \ifnum#1=\tw@      % only (re)new
2953       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2954       \bbl@tglobal\bbl@savetoday
2955       \bbl@tglobal\bbl@savestate
2956       \bbl@savestrings
2957     \fi
2958   \fi}

```

4.20. Processing keys in ini

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

```

2959 \def\bbl@inikv#1#2{%      key=value
2960   \toks@{#2}%             This hides #'s from ini values

```

```
2961 \bbl@csarg\edef{@kv@bbl@section.#1}{\the\toks@}}
```

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

```
2962 \let\bbl@inikv@identification\bbl@inikv
2963 \let\bbl@inikv@date\bbl@inikv
2964 \let\bbl@inikv@typography\bbl@inikv
2965 \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.

```
2966 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2967 \def\bbl@inikv@characters#1#2{%
2968   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2969   {\bbl@exp{%
2970     \\g@addto@macro\\bbl@release@casing{%
2971       \\bbl@casemapping{ }\languagename}{\unexpanded{#2}}}%
2972   {\in@{ $casing. }{ $#1 }% e.g., casing.Uv = uV
2973     \ifin@
2974       \lowercase{\def\bbl@tempb{#1}}%
2975       \bbl@replace\bbl@tempb{casing.}{ }%
2976       \bbl@exp{ \\g@addto@macro\\bbl@release@casing{%
2977         \\bbl@casemapping
2978         { \\bbl@maybextx\bbl@tempb }{\languagename}{\unexpanded{#2}}}%
2979       \else
2980         \bbl@inikv{#1}{#2}%
2981       \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’.

```
2982 \def\bbl@inikv@counters#1#2{%
2983   \bbl@ifsamestring{#1}{digits}%
2984   {\bbl@error{digits-is-reserved}{ }{ }{ }}%
2985   { }%
2986   \def\bbl@tempc{#1}%
2987   \bbl@trim@def{\bbl@tempb*}{#2}%
2988   \in@{.1$}{#1$}%
2989   \ifin@
2990     \bbl@replace\bbl@tempc{.1}{ }%
2991     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
2992       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2993   \fi
2994   \in@{.F.}{#1}%
2995   \ifin@\else\in@{.S.}{#1}\fi
2996   \ifin@
2997     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
2998   \else
2999     \toks@{ }% Required by \bbl@buildifcase, which returns \bbl@tempa
3000     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3001     \bbl@csarg\global\expandafter\let{cntr@#1@\languagename}\bbl@tempa
3002   \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.

```
3003 \ifcase\bbl@engine
3004   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3005     \bbl@ini@captions@aux{#1}{#2}}
3006 \else
3007   \def\bbl@inikv@captions#1#2{%
3008     \bbl@ini@captions@aux{#1}{#2}}
3009 \fi
```

The auxiliary macro for captions define `\caption`name.

```

3010 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3011 \bbl@replace\bbl@tempa{.template}{}%
3012 \def\bbl@toreplace{#1}{}%
3013 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3014 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3015 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3016 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname{}}%
3017 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3018 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3019 \ifin@
3020 \@nameuse{\bbl@patch\bbl@tempa}%
3021 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3022 \fi
3023 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3024 \ifin@
3025 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3026 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3027 \\\bbl@ifunset{\bbl@\bbl@tempa fmt@\\\language}%
3028 {[fnum@\bbl@tempa]}%
3029 {\\\@nameuse{\bbl@\bbl@tempa fmt@\\\language}}}%
3030 \fi}
3031 %
3032 \def\bbl@ini@captions@aux#1#2{%
3033 \bbl@trim@def\bbl@tempa{#1}%
3034 \bbl@xin@{.template}{\bbl@tempa}%
3035 \ifin@
3036 \bbl@ini@captions@template{#2}\language
3037 \else
3038 \bbl@ifblank{#2}%
3039 {\bbl@exp{%
3040 \toks@{\\\bbl@nocaption{\bbl@tempa name}\language\bbl@tempa name}}}%
3041 {\bbl@trim\toks@{#2}}%
3042 \bbl@exp{%
3043 \\\bbl@add\\bbl@savestrings{%
3044 \\\SetString\<\bbl@tempa name>\the\toks@}}}%
3045 \toks@\expandafter{\bbl@captionslist}%
3046 \bbl@exp{\\\in@{\<\bbl@tempa name>\the\toks@}}%
3047 \ifin@else
3048 \bbl@exp{%
3049 \\\bbl@add\<\bbl@extracaps@\language>\<\bbl@tempa name>}%
3050 \\\bbl@tglobal\<\bbl@extracaps@\language>}%
3051 \fi
3052 \fi}

```

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

```

3053 \def\bbl@list@the{%
3054 part,chapter,section,subsection,subsubsection,paragraph,%
3055 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3056 table,page,footnote,mpfootnote,mpfn}
3057 %
3058 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3059 \bbl@ifunset{\bbl@map@#1@\language}%
3060 {\@nameuse{#1}}%
3061 {\@nameuse{\bbl@map@#1@\language}}}
3062 %
3063 \def\bbl@map@lbl#1{% #1:a sign, eg, .
3064 \ifin@csname#1\else
3065 \bbl@ifunset{\bbl@map@#1@\language}%
3066 {#1}%
3067 {\@nameuse{\bbl@map@#1@\language}}%
3068 \fi}
3069 %

```

```

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

```

3125 \def\bbl@chapttype{chapter}
3126 \ifx\@makechapterhead\undefined
3127   \let\bbl@patchchapter\relax

```

```

3128 \else\ifx\thechapter\@undefined
3129 \let\bbl@patchchapter\relax
3130 \else\ifx\ps@headings\@undefined
3131 \let\bbl@patchchapter\relax
3132 \else
3133 \def\bbl@patchchapter{%
3134 \global\let\bbl@patchchapter\relax
3135 \gdef\bbl@chfmt{%
3136 \bbl@ifunset\bbl@\bbl@chapttype fmt@\language}%
3137 {\@chapapp\space\thechapter}%
3138 {\@nameuse\bbl@\bbl@chapttype fmt@\language}}}%
3139 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3140 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3141 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3142 \bbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3143 \bbl@tglobal\appendix
3144 \bbl@tglobal\ps@headings
3145 \bbl@tglobal\chaptermark
3146 \bbl@tglobal\makechapterhead}
3147 \let\bbl@patchappendix\bbl@patchchapter
3148 \fi\fi\fi
3149 \ifx\@part\@undefined
3150 \let\bbl@patchpart\relax
3151 \else
3152 \def\bbl@patchpart{%
3153 \global\let\bbl@patchpart\relax
3154 \gdef\bbl@partformat{%
3155 \bbl@ifunset\bbl@partfmt@\language}%
3156 {\partname\nobreakspace\thepart}%
3157 {\@nameuse\bbl@partfmt@\language}}}%
3158 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3159 \bbl@tglobal\@part}
3160 \fi

```

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

```

3161 \let\bbl@calendar\@empty
3162 \DeclareRobustCommand\localdate[1][\bbl@localdate{#1}]
3163 \def\bbl@localdate#1#2#3#4{%
3164 \begingroup
3165 \edef\bbl@they{#2}%
3166 \edef\bbl@them{#3}%
3167 \edef\bbl@thed{#4}%
3168 \edef\bbl@tempe{%
3169 \bbl@ifunset\bbl@calpr@\language}{\bbl@cl{calpr}},%
3170 #1}%
3171 \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3172 \bbl@replace\bbl@tempe{ }{}%
3173 \bbl@replace\bbl@tempe{convert}{convert=}%
3174 \let\bbl@ld@calendar\@empty
3175 \let\bbl@ld@variant\@empty
3176 \let\bbl@ld@convert\relax
3177 \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld@##1}{##2}}%
3178 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3179 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3180 \ifx\bbl@ld@calendar\@empty\else
3181 \ifx\bbl@ld@convert\relax\else
3182 \ifx\bbl@ld@convert\@empty
3183 \let\bbl@ld@convert\bbl@ld@calendar
3184 \else
3185 \edef\bbl@ld@convert{\expandafter\@gobble\bbl@ld@convert}%
3186 \fi
3187 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%

```

```

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

```

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).

```

3237 \let\bbl@calendar\@empty
3238 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3239     \bbl@xin@{\$calendrica:}{\$#2}%
3240     \ifin@
3241         \directlua{Babel.calendrica = Babel.calendrica or require("calendrica")}%
3242         \edef\bbl@tempa{\$#2}%
3243         \bbl@replace\bbl@tempa{\$calendrica:}{}%
3244         \bbl@replace\bbl@tempa{-}{_}%

```

```

3245 \bbl@afterelse
3246 \bbl@exp{\bbl@calendrica#1\@{\bbl@tempa}}%
3247 \else
3248 \bbl@afterfi
3249 \@nameuse{bbl@ca#2}#1\@
3250 \fi}
3251 \def\bbl@calendrica#1-#2-#3\@#4#5#6#7{%
3252 \directlua{
3253   local d = Babel.calendrica.new_date(
3254     {year=\number#1, month=\number#2, day=\number#3},
3255     Babel.calendrica.cal_gregorian)
3256   local r = Babel.calendrica.as_date(d, Babel.calendrica.cal_#4)
3257   token.set_macro('\bbl@stripslash#5', r.year)
3258   token.set_macro('\bbl@stripslash#6', r.month)
3259   token.set_macro('\bbl@stripslash#7', r.day) }}
3260 \newcommand\BabelDateSpace{\nobreakspace}
3261 \newcommand\BabelDateDot{.\@}
3262 \newcommand\BabelDated[1]{\number#1}
3263 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3264 \newcommand\BabelDateM[1]{\number#1}
3265 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3266 \newcommand\BabelDateMMMM[1]{%
3267   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3268 \newcommand\BabelDatey[1]{\number#1}%
3269 \newcommand\BabelDateyy[1]{%
3270   \ifnum#1<10 0\number#1 %
3271   \else\ifnum#1<100 \number#1 %
3272   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3273   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3274   \else
3275     \bbl@error{limit-two-digits}{\number#1}%
3276   \fi\fi\fi\fi}
3277 \newcommand\BabelDateyyyy[1]{\number#1}
3278 \newcommand\BabelDateU[1]{\number#1}%
3279 \def\bbl@replace@finish@iii#1{%
3280   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3281 \def\bbl@TG@date{%
3282   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3283   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot}}%
3284   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3285   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr{###1}}%
3286   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3287   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3288   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3289   \bbl@replace\bbl@toreplace{[M]}{\bbl@datecctr{###2}}%
3290   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3291   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMMM{###2}}%
3292   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3293   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr{###3}}%
3294   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3295   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3296   \bbl@replace\bbl@toreplace{[U]}{\bbl@datecctr{###1}}%
3297   \bbl@replace@finish@iii\bbl@toreplace}
3298 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3299 \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.

```

3300 \AddToHook{begindocument/before}{%
3301   \let\bbl@normalsf\normalsfcodes
3302   \let\normalsfcodes\relax}

```

```

3303 \AtBeginDocument{%
3304   \ifx\bbl@normalsf\@empty
3305     \ifnum\sfcode`\.=\@m
3306       \let\normalsfcodes\frenchspacing
3307     \else
3308       \let\normalsfcodes\nonfrenchspacing
3309     \fi
3310   \else
3311     \let\normalsfcodes\bbl@normalsf
3312   \fi}

```

Transforms.

Process the transforms read from ini files, converts them to a form close to the user interface (with `\bbl@prehyphenation` and `\bbl@prehyphenation`), 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.

```

3313 \bbl@csarg\let\inikv@transforms.prehyphenation\bbl@inikv
3314 \bbl@csarg\let\inikv@transforms.posthyphenation\bbl@inikv
3315 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3316   #1{#2}{#3}{#4}{#5}}
3317 \beginingroup
3318   \catcode`\%=12
3319   \catcode`\&=14
3320   \gdef\bbl@transforms#1#2#3{%&
3321     \directlua{
3322       local str = [==[#2]==]
3323       str = str:gsub('%.%d+%.%d+$', '')
3324       token.set_macro('babeltempa', str)
3325     }&
3326     \def\babeltempc{}&
3327     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3328     \ifin@ \else
3329       \bbl@xin@{:\babeltempa,}{,\bbl@KVP@transforms,}&
3330     \fi
3331     \ifin@
3332       \bbl@foreach\bbl@KVP@transforms{%&
3333         \bbl@xin@{:\babeltempa,}{,##1,}&
3334         \ifin@ & font:font:transform syntax
3335           \directlua{
3336             local t = {}
3337             for m in string.gmatch('#1'..' ':'(.)') do
3338               table.insert(t, m)
3339             end
3340             table.remove(t)
3341             token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3342           }&
3343         \fi&
3344       \in@{.0$}{#2$}&
3345       \ifin@
3346         \directlua{%& (\attribute) syntax
3347           local str = string.match([\bbl@KVP@transforms],
3348             '%(([^()^-)%(^)]-\babeltempa)')
3349           if str == nil then
3350             token.set_macro('babeltempb', '')
3351           else
3352             token.set_macro('babeltempb', ',attribute=' .. str)
3353           end
3354         }&
3355         \toks@{#3}&
3356         \bbl@exp{%&
3357           \\g@addto@macro\\bbl@release@transforms{%&
3358             \relax & Closes previous \bbl@transforms@aux

```

```

3359          \\bbl@transforms@aux
3360          \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3361          {\language}\the\toks@}}&%
3362      \else
3363          \g@addto@macro\bbl@release@transforms{, {#3}}&%
3364      \fi
3365  \fi}
3366 \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.

```

3367 \def\bbl@provide@lsys#1{%
3368   \bbl@ifunset{bbl@lname@#1}%
3369   {\bbl@load@info{#1}}%
3370   }%
3371   \bbl@csarg\let{lsys@#1}\empty
3372   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3373   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3374   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3375   \bbl@ifunset{bbl@lname@#1}{}%
3376   {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3377   \ifcase\bbl@engine\or\or
3378   \bbl@ifunset{bbl@prehc@#1}{}%
3379   {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3380    }%
3381    {\ifx\bbl@xenohyph\undefined
3382     \global\let\bbl@xenohyph\bbl@xenohyph@d
3383     \ifx\AtBeginDocument\@notprerr
3384       \expandafter\@secondoftwo % to execute right now
3385     \fi
3386     \AtBeginDocument{%
3387       \bbl@patchfont{\bbl@xenohyph}%
3388       {\expandafter\select@language\expandafter{\language}}}%
3389   \fi}}%
3390 \fi
3391 \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.

```

3392 \def\bbl@setdigits#1#2#3#4#5{%
3393   \bbl@exp{%
3394     \def\<\language name digits>###1{% i.e., \langdigits
3395       \<bbl@digits@\language name>###1\\\@nil}%
3396       \let\<bbl@cntr@digits@\language name>\<\language name digits>%
3397       \def\<\language name counter>###1{% i.e., \langcounter
3398         \\\expandafter\<bbl@counter@\language name>%
3399         \\\csname c@###1\endcsname}%
3400       \def\<bbl@counter@\language name>###1% i.e., \bbl@counter@lang
3401         \\\expandafter\<bbl@digits@\language name>%
3402         \\\number###1\\\@nil}%
3403   \def\bbl@tempa##1##2##3##4##5{%
3404     \bbl@exp{% Wow, quite a lot of hashes! :-(
3405       \def\<bbl@digits@\language name>#####1{%
3406         \\\ifx#####1\\\@nil % i.e., \bbl@digits@lang
3407         \\\else

```

[illegible]

```

3423 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={
3424   \ifx\##1%                % \ before, in case #1 is multiletter
3425     \bbl@exp{%
3426       \def\\bbl@tempa####1{%
3427         \<ifcase>####1space\the\toks@\<else>\\@ctrerr\<fi>}}%
3428   \else
3429     \toks@\expandafter{\the\toks@\or #1}%
3430     \expandafter\bbl@buildifcase
3431   \fi}

```

```

3432 \newcommand\localenumerical[2]{%
3433   \bbl@ifunset{\bbl@cntr@#1@\language}%
3434     {#2}%
3435     {\bbl@cs{cntr@#1@\language}{#2}}}
3436 \def\bbl@localecntr#1#2{\localenumerical{#2}{#1}}
3437 \newcommand\localecounter[2]{%
3438   \expandafter\bbl@localecntr
3439   \expandafter{\number\csname c@#2\endcsname}{#1}}
3440 \def\bbl@alphnumerical#1#2{%
3441   \expandafter\bbl@alphnumerical@i\number#2 76543210\@@{#1}}
3442 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
3443   \ifcase\car#8\@nil\or    % Currently <10000, but prepared for bigger
3444     \bbl@alphnumerical@ii{#9}000000#1\or
3445     \bbl@alphnumerical@ii{#9}00000#1#2\or
3446     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3447     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3448     \bbl@alphnum@invalid{>9999}%
3449   \fi}
3450 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
3451   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3452     {\bbl@cs{cntr@#1.4@\language}{#5}%
3453     \bbl@cs{cntr@#1.3@\language}{#6}%
3454     \bbl@cs{cntr@#1.2@\language}{#7}%
3455     \bbl@cs{cntr@#1.1@\language}{#8}%
3456     \ifnum#6#7#8>\z@
3457       \bbl@ifunset{\bbl@cntr@#1.S.321@\language}{}%
3458       {\bbl@cs{cntr@#1.S.321@\language}{}%
3459       \fi}%
3460     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}
3461 \def\bbl@alphnum@invalid#1{%
3462   \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```
3463 \newcommand\BabelUppercaseMapping[3]{%
3464   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3465 \newcommand\BabelTitlecaseMapping[3]{%
3466   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3467 \newcommand\BabelLowercaseMapping[3]{%
3468   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

  The parser for casing and casing. (variant).
3469 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3470   \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3471 \else
3472   \def\bbl@uftocode#1{\expandafter\string#1}
3473 \fi
3474 \def\bbl@casemapping#1#2#3{% 1:variant
3475   \def\bbl@tempa##1 ##2{% Loop
3476     \bbl@casemapping@i{##1}%
3477     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3478   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3479   \def\bbl@tempe{0}% Mode (upper/lower...)
3480   \def\bbl@tempc{#3}% Casing list
3481   \expandafter\bbl@tempa\bbl@tempc\@empty}
3482 \def\bbl@casemapping@i#1{%
3483   \def\bbl@tempb{#1}%
3484   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3485     \@nameuse{regex_replace_all:nnN}%
3486     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\{\0}\}\bbl@tempb
3487   \else
3488     \@nameuse{regex_replace_all:nnN}{.}{\{\0}\}\bbl@tempb
3489   \fi
3490   \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3491 \def\bbl@casemapping@ii#1#2#3\@@{%
3492   \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3493   \ifin@
3494     \edef\bbl@tempe{%
3495       \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3496   \else
3497     \ifcase\bbl@tempe\relax
3498       \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3499       \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3500     \or
3501       \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3502     \or
3503       \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3504     \or
3505       \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3506   \fi
3507 \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.

```
3508 \def\bbl@localeinfo#1#2{%
3509   \bbl@ifunset{bbl@info@#2}{#1}%
3510   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}{#1}%
3511    {\bbl@cs{csname bbl@info@#2\endcsname @\language}}}%
3512 \newcommand\localeinfo[1]{%
3513   \ifx*#1\@empty
3514     \bbl@afterelse\bbl@localeinfo{}%
3515   \else
3516     \bbl@localeinfo
3517     {\bbl@error{no-ini-info}}{\{\}\{\}}%
3518     {#1}%
3519   \fi}
```

```

3519 \fi}
3520 % \@namedef{bbl@info@name.locale}{\lname}
3521 \@namedef{bbl@info@tag.ini}{\lini}
3522 \@namedef{bbl@info@name.english}{\elname}
3523 \@namedef{bbl@info@name.opentype}{\lname}
3524 \@namedef{bbl@info@tag.bcp47}{\tbc}
3525 \@namedef{bbl@info@language.tag.bcp47}{\lbc}
3526 \@namedef{bbl@info@tag.opentype}{\lotf}
3527 \@namedef{bbl@info@script.name}{\esname}
3528 \@namedef{bbl@info@script.name.opentype}{\sname}
3529 \@namedef{bbl@info@script.tag.bcp47}{\sbc}
3530 \@namedef{bbl@info@script.tag.opentype}{\sotf}
3531 \@namedef{bbl@info@region.tag.bcp47}{\rbcp}
3532 \@namedef{bbl@info@variant.tag.bcp47}{\vbcp}
3533 \@namedef{bbl@info@extension.t.tag.bcp47}{\extt}
3534 \@namedef{bbl@info@extension.u.tag.bcp47}{\extu}
3535 \@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.

```

3536 << *More package options >> ≡
3537 \DeclareOption{ensureinfo=off}{}
3538 << /More package options >>
3539 \let\BabelEnsureInfo\relax

```

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

```

3540 \newcommand\getLocaleproperty{%
3541   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3542 \def\bbl@getproperty@s#1#2#3{%
3543   \let#1\relax
3544   \def\bbl@elt##1##2##3{%
3545     \bbl@ifsamestring{##1/##2}{#3}%
3546     {\providecommand#1{##3}%
3547     \def\bbl@elt####1####2####3{}}}%
3548   {}}%
3549   \bbl@cs{inidata@#2}}%
3550 \def\bbl@getproperty@x#1#2#3{%
3551   \bbl@getproperty@s{#1}{#2}{#3}%
3552   \ifx#1\relax
3553     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3554   \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.

```

3555 \let\bbl@ini@loaded\@empty
3556 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3557 \def\ShowLocaleProperties#1{%
3558   \typeout{}}%
3559   \typeout{*** Properties for language '#1' ***}%
3560   \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3561   \@nameuse{\bbl@inidata@#1}%
3562   \typeout{*****}}

```

4.26. BCP 47 related commands

This macro is called by language selectors when the language isn't recognized. So, it's the core for (1) mapping from a BCP 47 tag to the actual language, if bcp47.toname is enabled (i.e., if bbl@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.

```

3563 \newif\ifbbl@bcppallowed

```

```

3564 \bbl@bcpallowedfalse
3565 \def\bbl@autoload@options{@import}
3566 \def\bbl@provide@locale{%
3567   \ifx\babelprovide\undefined
3568     \bbl@error{base-on-the-fly}{}}}%
3569 \fi
3570 \let\bbl@auxname\language
3571 \ifbbl@bcptoname
3572   \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
3573   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
3574   \let\locale\language}%
3575 \fi
3576 \ifbbl@bcpallowed
3577   \expandafter\ifx\csname date\language\endcsname\relax
3578     \expandafter
3579     \bbl@bcplookup{\language}%-\@empty-\@empty-\@empty\@@
3580     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3581       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3582       \let\locale\language
3583       \expandafter\ifx\csname date\language\endcsname\relax
3584         \let\bbl@initoload\bbl@bcp
3585         \bbl@exp{\\babelprovide[\bbl@autoload@bcptoptions]{\language}}}%
3586         \let\bbl@initoload\relax
3587       \fi
3588       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\locale}%
3589     \fi
3590   \fi
3591 \fi
3592 \expandafter\ifx\csname date\language\endcsname\relax
3593   \IfFileExists{babel-\language.tex}%
3594   {\bbl@exp{\\babelprovide[\bbl@autoload@options]{\language}}}%
3595   {}%
3596 \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`.

```

3597 \providecommand\BCPdata{}
3598 \ifx\renewcommand\undefined\else
3599   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3600   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3601     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3602     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3603     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3604   \def\bbl@bcpdata@ii#1#2{%
3605     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3606     {\bbl@error{unknown-ini-field}{#1}{}}}%
3607     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3608     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3609 \fi
3610 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3611 \@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.

```

3612 \newcommand\babeladjust[1]{%
3613   \bbl@forkv{#1}{%
3614     \bbl@ifunset{bbl@ADJ@##1@##2}%
3615     {\bbl@cs{ADJ@##1}{##2}}}%

```

```

3616      {\bbl@cs{ADJ@##1@##2}}}}
3617 %
3618 \def\bbl@adjust@lua#1#2{%
3619   \ifvmode
3620     \ifnum\currentgrouplevel=\z@
3621       \directlua{ Babel.#2 }%
3622       \expandafter\expandafter\expandafter\@gobble
3623     \fi
3624   \fi
3625   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3626 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3627   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3628 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3629   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3630 \@namedef{\bbl@ADJ@bidi.text@on}{%
3631   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3632 \@namedef{\bbl@ADJ@bidi.text@off}{%
3633   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3634 \@namedef{\bbl@ADJ@bidi.math@on}{%
3635   \let\bbl@noamsmath\empty}
3636 \@namedef{\bbl@ADJ@bidi.math@off}{%
3637   \let\bbl@noamsmath\relax}
3638 %
3639 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3640   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3641 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3642   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3643 %
3644 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3645   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3646 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3647   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3648 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
3649   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3650 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
3651   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3652 \@namedef{\bbl@ADJ@justify.arabic@on}{%
3653   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3654 \@namedef{\bbl@ADJ@justify.arabic@off}{%
3655   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3656 %
3657 \def\bbl@adjust@layout#1{%
3658   \ifvmode
3659     #1%
3660     \expandafter\@gobble
3661   \fi
3662   {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3663 \@namedef{\bbl@ADJ@layout.tabular@on}{%
3664   \ifnum\bbl@tabular@mode=\tw@
3665     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3666   \else
3667     \chardef\bbl@tabular@mode\@ne
3668   \fi}
3669 \@namedef{\bbl@ADJ@layout.tabular@off}{%
3670   \ifnum\bbl@tabular@mode=\tw@
3671     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3672   \else
3673     \chardef\bbl@tabular@mode\z@
3674   \fi}
3675 \@namedef{\bbl@ADJ@layout.lists@on}{%
3676   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3677 \@namedef{\bbl@ADJ@layout.lists@off}{%
3678   \bbl@adjust@layout{\let\list\bbl@OL@list}}

```

```

3679 %
3680 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3681   \bbl@bcpallowedtrue}
3682 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3683   \bbl@bcpallowedfalse}
3684 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3685   \def\bbl@bcp@prefix{#1}}
3686 \def\bbl@bcp@prefix{bcp47-}
3687 \@namedef{bbl@ADJ@autoload.options}#1{%
3688   \def\bbl@autoload@options{#1}}
3689 \def\bbl@autoload@bcptoptions{import}
3690 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3691   \def\bbl@autoload@bcptoptions{#1}}
3692 \newif\ifbbl@bcptoname
3693 %
3694 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3695   \bbl@bcptonametrue}
3696 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3697   \bbl@bcptonamefalse}
3698 %
3699 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3700   \directlua{ Babel.ignore_pre_char = function(node)
3701     return (node.lang == \the\csname l@nohyphenation\endcsname)
3702   end }}
3703 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3704   \directlua{ Babel.ignore_pre_char = function(node)
3705     return false
3706   end }}
3707 %
3708 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3709   \def\bbl@ignoreinterchar{%
3710     \ifnum\language=\l@nohyphenation
3711       \expandafter\@gobble
3712     \else
3713       \expandafter\@firstofone
3714     \fi}}
3715 \@namedef{bbl@ADJ@interchar.disable@off}{%
3716   \let\bbl@ignoreinterchar\@firstofone}
3717 %
3718 \@namedef{bbl@ADJ@select.write@shift}{%
3719   \let\bbl@restorelastskip\relax
3720   \def\bbl@savelastskip{%
3721     \let\bbl@restorelastskip\relax
3722     \ifvmode
3723       \ifdim\lastskip=\z@
3724         \let\bbl@restorelastskip\nobreak
3725       \else
3726         \bbl@exp{%
3727           \def\\bbl@restorelastskip{%
3728             \skip@=\the\lastskip
3729             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3730         \fi
3731       \fi}}
3732 \@namedef{bbl@ADJ@select.write@keep}{%
3733   \let\bbl@restorelastskip\relax
3734   \let\bbl@savelastskip\relax}
3735 \@namedef{bbl@ADJ@select.write@omit}{%
3736   \AddBabelHook{babel-select}{beforestart}{%
3737     \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3738   \let\bbl@restorelastskip\relax
3739   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3740 \@namedef{bbl@ADJ@select.encoding@off}{%
3741   \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.

```
3742 << *More package options >> ≡
3743 \DeclareOption{safe=none}{\let\bbl@opt@safe\empty}
3744 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3745 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3746 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3747 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3748 << /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.

```
3749 \bbl@trace{Cross referencing macros}
3750 \ifx\bbl@opt@safe\empty\else % i.e., if 'ref' and/or 'bib'
3751   \def\@newl@bel#1#2#3{%
3752     {\@safe@activestrue
3753       \bbl@ifunset{#1@#2}%
3754         \relax
3755       {\gdef\@multiplelabels{%
3756         \@latex@warning@no@line{There were multiply-defined labels}}%
3757         \@latex@warning@no@line{Label `#2' multiply defined}}%
3758       \global\@namedef{#1@#2}{#3}}}
```

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

```
3759 \CheckCommand*\@testdef[3]{%
3760   \def\reserved@a{#3}%
3761   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3762   \else
3763     \@tempwattrue
3764   \fi}
```

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

```
3765 \def\@testdef#1#2#3{%
3766   \@safe@activestrue
3767   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3768   \def\bbl@tempb{#3}%
3769   \@safe@activesfalse
3770   \ifx\bbl@tempa\relax
3771   \else
3772     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3773   \fi
3774   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3775   \ifx\bbl@tempa\bbl@tempb
3776   \else
3777     \@tempwattrue
3778   \fi}
3779 \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.

```
3780 \bbl@xin@{R}\bbl@opt@safe
3781 \ifin@
3782 \edef\bbl@tempc{\expandafter\string\csname ref_code\endcsname}%
3783 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3784 {\expandafter\strip@prefix\meaning\ref}%
3785 \ifin@
3786 \bbl@redefine\@kernel@ref#1{%
3787   \@safe@activetrue\org@@kernel@ref{#1}\@safe@activfalse}
3788 \bbl@redefine\@kernel@pageref#1{%
3789   \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activfalse}
3790 \bbl@redefine\@kernel@sref#1{%
3791   \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3792 \bbl@redefine\@kernel@spageref#1{%
3793   \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3794 \else
3795 \bbl@redefineroobust\ref#1{%
3796   \@safe@activetrue\org@ref{#1}\@safe@activfalse}
3797 \bbl@redefineroobust\pageref#1{%
3798   \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
3799 \fi
3800 \else
3801 \let\org@ref\ref
3802 \let\org@pageref\pageref
3803 \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.

```
3804 \bbl@xin@{B}\bbl@opt@safe
3805 \ifin@
3806 \bbl@redefine\@citex[#1]#2{%
3807   \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activfalse
3808   \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.)

```
3809 \AtBeginDocument{%
3810   \@ifpackageloaded{natbib}{%
3811     \def\@citex[#1][#2]#3{%
3812       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activfalse
3813       \org@@citex[#1][#2]{\bbl@tempa}}%
3814   }}
```

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

```
3815 \AtBeginDocument{%
3816   \@ifpackageloaded{cite}{%
3817     \def\@citex[#1]#2{%
3818       \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3819   }}
```

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

```
3820 \bbl@redefine\nocite#1{%
3821   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3822 \bbl@redefine\bibcite{%
3823   \bbl@cite@choice
3824   \bibcite}
```

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

```
3825 \def\bbl@bibcite#1#2{%
3826   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3827 \def\bbl@cite@choice{%
3828   \global\let\bibcite\bbl@bibcite
3829   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3830   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3831   \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.

```
3832 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3833 \bbl@redefine\@bibitem#1{%
3834   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3835 \else
3836   \let\org@nocite\nocite
3837   \let\org@@citex\@citex
3838   \let\org@bibcite\bibcite
3839   \let\org@@bibitem\@bibitem
3840 \fi
```

5.2. Layout

```
3841 \newcommand\BabelPatchSection[1]{%
3842   \@ifundefined{#1}{}{%
3843     \bbl@exp{\let\bbl@ss@#1>\<#1>}%
3844     \@namedef{#1}{%
3845       \@ifstar{\bbl@presec@s{#1}}%
3846       {\@dblarg{\bbl@presec@x{#1}}}}}%
3847 \def\bbl@presec@x#1[#2]#3{%
3848   \bbl@exp{%
3849     \\select@language@x{\bbl@main@language}%
3850     \\bbl@cs{sspre@#1}%
3851     \\bbl@cs{ss@#1}%
3852     [\\foreignlanguage{\language}\unexpanded{#2}}}%
3853     {\\foreignlanguage{\language}\unexpanded{#3}}}%
3854   \\select@language@x{\language}}}
```

```

3855 \def\bbl@presec@s#1#2{%
3856   \bbl@exp{%
3857     \select@language{x{\bbl@main@language}%
3858     \bbl@cs{sspre@#1}%
3859     \bbl@cs{ss@#1}*\%
3860     {\foreignlanguage{\language}{\unexpanded{#2}}}%
3861     \select@language{x{\language}}}
3862 %
3863 \IfBabelLayout{sectioning}%
3864   {\BabelPatchSection{part}%
3865   \BabelPatchSection{chapter}%
3866   \BabelPatchSection{section}%
3867   \BabelPatchSection{subsection}%
3868   \BabelPatchSection{subsubsection}%
3869   \BabelPatchSection{paragraph}%
3870   \BabelPatchSection{subparagraph}%
3871   \def\babel@toc#1{%
3872     \select@language{x{\bbl@main@language}}}{%
3873 \IfBabelLayout{captions}%
3874   {\BabelPatchSection{caption}}}{%

```

\BabelFootnote Footnotes.

```

3875 \bbl@trace{Footnotes}
3876 \def\bbl@footnote#1#2#3{%
3877   \ifnextchar[%
3878     {\bbl@footnote@o{#1}{#2}{#3}}%
3879     {\bbl@footnote@x{#1}{#2}{#3}}
3880 \long\def\bbl@footnote@x#1#2#3#4{%
3881   \bgroup
3882     \select@language{x{\bbl@main@language}%
3883     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3884   \egroup}
3885 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3886   \bgroup
3887     \select@language{x{\bbl@main@language}%
3888     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3889   \egroup}
3890 \def\bbl@footnotetext#1#2#3{%
3891   \ifnextchar[%
3892     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3893     {\bbl@footnotetext@x{#1}{#2}{#3}}
3894 \long\def\bbl@footnotetext@x#1#2#3#4{%
3895   \bgroup
3896     \select@language{x{\bbl@main@language}%
3897     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3898   \egroup}
3899 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3900   \bgroup
3901     \select@language{x{\bbl@main@language}%
3902     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3903   \egroup}
3904 \def\BabelFootnote#1#2#3#4{%
3905   \ifx\bbl@fn@footnote\undefined
3906     \let\bbl@fn@footnote\footnote
3907   \fi
3908   \ifx\bbl@fn@footnotetext\undefined
3909     \let\bbl@fn@footnotetext\footnotetext
3910   \fi
3911   \bbl@iifblank{#2}%
3912     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3913     \namedef{\bbl@stripslash#1text}%
3914     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3915   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%

```

```

3916 \namedef{\bbl@stripslash#1text}%
3917 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}%
3918 \IfBabelLayout{footnotes}%
3919 {\let\bbl@OL@footnote\footnote
3920 \BabelFootnote\footnote\languagename{}}%
3921 \BabelFootnote\localfootnote\languagename{}}%
3922 \BabelFootnote\mainfootnote{}}%
3923 {}

```

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.

```

3924 \bbl@trace{Marks}
3925 \IfBabelLayout{sectioning}
3926 {\ifx\bbl@opt@headfoot\@nnil
3927 \g@addto@macro\resetactivechars{%
3928 \set@typeset@protect
3929 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3930 \let\protect\noexpand
3931 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3932 \edef\thepage{%
3933 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3934 \fi}%
3935 \fi}
3936 {\ifbbl@single\else
3937 \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3938 \markright#1}%
3939 \bbl@ifblank{#1}%
3940 {\org@markright{}}%
3941 {\toks@{#1}%
3942 \bbl@exp{%
3943 \org@markright{\protect\foreignlanguage{\languagename}%
3944 \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, ~~La~~TeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3945 \ifx\@mkboth\markboth
3946 \def\bbl@tempc{\let\@mkboth\markboth}%
3947 \else
3948 \def\bbl@tempc{}%
3949 \fi
3950 \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3951 \markboth#1#2{%
3952 \protected@edef\bbl@tempb##1{%
3953 \protect\foreignlanguage
3954 {\languagename}{\protect\bbl@restore@actives##1}}%
3955 \bbl@ifblank{#1}%
3956 {\toks@{}}%
3957 {\toks@\expandafter{\bbl@tempb{#1}}}%
3958 \bbl@ifblank{#2}%
3959 {\@temptokena{}}%
3960 {\@temptokena\expandafter{\bbl@tempb{#2}}}%

```

```

3961      \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}}%
3962      \bbl@tempc
3963      \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.

```

3964 \bbl@trace{Preventing clashes with other packages}
3965 \ifx\org@ref\undefined\else
3966   \bbl@xin@{R}\bbl@opt@safe
3967   \ifin@
3968     \AtBeginDocument{%
3969       \@ifpackageloaded{ifthen}{%
3970         \bbl@redefine@long\ifthenelse#1#2#3{%
3971           \let\bbl@temp@pref\pageref
3972           \let\pageref\org@pageref
3973           \let\bbl@temp@ref\ref
3974           \let\ref\org@ref
3975           \@safe@activestrue
3976           \org@ifthenelse{#1}%
3977             {\let\pageref\bbl@temp@pref
3978              \let\ref\bbl@temp@ref
3979              \@safe@activesfalse
3980              #2}%
3981             {\let\pageref\bbl@temp@pref
3982              \let\ref\bbl@temp@ref
3983              \@safe@activesfalse
3984              #3}%
3985           }%
3986         }{}%
3987       }
3988 \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`.

```

3989 \AtBeginDocument{%
3990   \@ifpackageloaded{varioref}{%
3991     \bbl@redefine\@@vpageref#1[#2]#3{%
3992       \@safe@activestrue
3993       \org@@@vpageref{#1}[#2]{#3}%

```

```

3994      \@safe@activesfalse}%
3995      \bbl@redefine\hrefpagenum#1#2{%
3996      \@safe@activetrue
3997      \org@hrefpagenum{#1}{#2}%
3998      \@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.

```

3999      \expandafter\def\csname Ref \endcsname#1{%
4000      \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
4001      }{}%
4002  }
4003 \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.

```

4004 \AtEndOfPackage{%
4005   \AtBeginDocument{%
4006     \@ifpackageloaded{hhline}%
4007     {\expandafter\ifx\csname normal@char\string\endcsname\relax
4008     \else
4009       \makeatletter
4010       \def\@currname{hhline}\input{hhline.sty}\makeatother
4011       \fi}%
4012     {}}}

```

`\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`).

```

4013 \def\substitutefontfamily#1#2#3{%
4014   \lowercase{\immediate\openout15=#1#2.fd\relax}%
4015   \immediate\write15{%
4016     \string\ProvidesFile{#1#2.fd}%
4017     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
4018     \space generated font description file]^J
4019     \string\DeclareFontFamily{#1}{#2}{}^J
4020     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^J
4021     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^J
4022     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^J
4023     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^J
4024     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^J
4025     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^J
4026     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^J
4027     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^J
4028   }%
4029   \closeout15
4030 }
4031 \@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

```

4032 \bbl@trace{Encoding and fonts}
4033 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
4034 \newcommand\BabelNonText{TS1,T3,TS3}
4035 \let\org@TeX\TeX
4036 \let\org@LaTeX\LaTeX
4037 \let\ensureascii@firstofone
4038 \let\asciientcoding\@empty
4039 \AtBeginDocument{%
4040   \def\@elt#1{,#1,}%
4041   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4042   \let\@elt\relax
4043   \let\bbl@tempb\@empty
4044   \def\bbl@tempc{OT1}%
4045   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
4046     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
4047   \bbl@foreach\bbl@tempa{%
4048     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
4049     \ifin@
4050       \def\bbl@tempb{#1}% Store last non-ascii
4051     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
4052       \ifin@
4053         \def\bbl@tempc{#1}% Store last ascii
4054       \fi
4055     \fi}%
4056   \ifx\bbl@tempb\@empty\else
4057     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
4058     \ifin@
4059       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
4060     \fi
4061     \let\asciientcoding\bbl@tempc
4062     \renewcommand\ensureascii[1]{%
4063       {\fontencoding{\asciientcoding}\selectfont#1}}%
4064     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
4065     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
4066   \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.

```

4067 \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.

```

4068 \AtBeginDocument{%
4069   \@ifpackageloaded{fontspec}%
4070   {\xdef\latinencoding{%
4071     \ifx\UTFencname\undefined
4072       EU\ifcase\bbl@engine\or2\or1\fi
4073     \else
4074       \UTFencname
4075     \fi}}%
4076   {\gdef\latinencoding{OT1}%
4077     \ifx\cf@encoding\bbl@t@one
4078       \xdef\latinencoding{\bbl@t@one}%

```

```

4079 \else
4080 \def\elt#1{,#1,}%
4081 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4082 \let\elt\relax
4083 \bbl@xin@{,Tl,}\bbl@tempa
4084 \ifin@
4085 \xdef\latinencoding{\bbl@t@one}%
4086 \fi
4087 \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.

```

4088 \DeclareRobustCommand{\latintext}{%
4089 \fontencoding{\latinencoding}\selectfont
4090 \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.

```

4091 \ifx\@undefined\DeclareTextFontCommand
4092 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
4093 \else
4094 \DeclareTextFontCommand{\textlatin}{\latintext}
4095 \fi

```

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

```

4096 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.6. Basic bidi support

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

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

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

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

```

4097 \bbl@trace{Loading basic (internal) bidi support}
4098 \ifodd\bbl@engine
4099 \else % Any xe+lua bidi
4100 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
4101 \bbl@error{bidi-only-lua}{\}\}\}%
4102 \let\bbl@beforeforeign\leavevmode
4103 \AtEndOfPackage{%
4104 \EnableBabelHook{babel-bidi}%
4105 \bbl@xebidipar}
4106 \fi\fi
4107 \def\bbl@loadxebidi#1{%

```

```

4108 \ifx\RTLfootnotetext\@undefined
4109 \AtEndOfPackage{%
4110 \EnableBabelHook{babel-bidi}%
4111 \ifx\fontspec\@undefined
4112 \usepackage{fontspec}% bidi needs fontspec
4113 \fi
4114 \usepackage#1{bidi}%
4115 \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4116 \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4117 \ifnum\@nameuse{bbl@wdir@languagename}=\tw@ % 'AL' bidi
4118 \bbl@digitsdotdash % So ignore in 'R' bidi
4119 \fi}}%
4120 \fi}
4121 \ifnum\bbl@bidimode>200 % Any xe bidi=
4122 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4123 \bbl@tentative{bidi=bidi}
4124 \bbl@loadxebidi{}
4125 \or
4126 \bbl@loadxebidi{[rldocument]}
4127 \or
4128 \bbl@loadxebidi{}
4129 \fi
4130 \fi
4131 \fi
4132 \ifnum\bbl@bidimode=\@ne % bidi=default
4133 \let\bbl@beforeforeign\leavevmode
4134 \ifodd\bbl@engine % lua
4135 \newattribute\bbl@attr@dir
4136 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4137 \bbl@exp{\output{\bodydir\pagedir\the\output}}
4138 \fi
4139 \AtEndOfPackage{%
4140 \EnableBabelHook{babel-bidi}% pdf/lua/xe
4141 \ifodd\bbl@engine\else % pdf/xe
4142 \bbl@xebidipar
4143 \fi}
4144 \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.

```

4145 \bbl@trace{Macros to switch the text direction}
4146 \def\bbl@alscripts{%
4147 ,Arabic,Syriac,Thaana,Hanifi_Rohingya,Hanifi,Sogdian,}
4148 \def\bbl@rscripts{%
4149 Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4150 Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
4151 Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
4152 Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
4153 Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
4154 Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
4155 Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
4156 Meroitic,N'Ko,Orkhon,Todhri}
4157 %
4158 \def\bbl@provide@dirs#1{%
4159 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4160 \ifin@
4161 \global\bbl@csarg\chardef{wdir@#1}\@ne
4162 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4163 \ifin@
4164 \global\bbl@csarg\chardef{wdir@#1}\tw@
4165 \fi
4166 \else

```

```

4167 \global\bbbl@csarg\chardef{wdir@#1}\z@
4168 \fi
4169 \ifodd\bbbl@engine
4170 \bbbl@csarg\ifcase{wdir@#1}%
4171 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
4172 \or
4173 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
4174 \or
4175 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
4176 \fi
4177 \fi}
4178 %
4179 \def\bbbl@switchdir{%
4180 \bbbl@ifunset{bbbl@lsys@\language name}{\bbbl@provide@lsys{\language name}}{}%
4181 \bbbl@ifunset{bbbl@wdir@\language name}{\bbbl@provide@dirs{\language name}}{}%
4182 \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}}%
4183 \def\bbbl@setdirs#1{%
4184 \ifcase\bbbl@select@type
4185 \bbbl@bodydir{#1}%
4186 \bbbl@pardir{#1}% <- Must precede \bbbl@texmdir
4187 \fi
4188 \bbbl@texmdir{#1}}
4189 \ifnum\bbbl@bidimode>\z@
4190 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4191 \DisableBabelHook{babel-bidi}
4192 \fi

```

Now the engine-dependent macros.

```

4193 \ifodd\bbbl@engine % luatex=1
4194 \else % pdftex=0, xetex=2
4195 \newcount\bbbl@dirlevel
4196 \chardef\bbbl@thetexmdir\z@
4197 \chardef\bbbl@thepardir\z@
4198 \def\bbbl@texmdir#1{%
4199 \ifcase#1\relax
4200 \chardef\bbbl@thetexmdir\z@
4201 \@nameuse{setlatin}%
4202 \bbbl@texmdir@i\beginL\endL
4203 \else
4204 \chardef\bbbl@thetexmdir\@ne
4205 \@nameuse{setnonlatin}%
4206 \bbbl@texmdir@i\beginR\endR
4207 \fi}
4208 \def\bbbl@texmdir@i#1#2{%
4209 \ifhmode
4210 \ifnum\currentgrouplevel>\z@
4211 \ifnum\currentgrouplevel=\bbbl@dirlevel
4212 \bbbl@error{multiple-bidi}{}{}%
4213 \bgroup\aftergroup#2\aftergroup\egroup
4214 \else
4215 \ifcase\currentgrouptype\or % 0 bottom
4216 \aftergroup#2% 1 simple {}
4217 \or
4218 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4219 \or
4220 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4221 \or\or\or % vbox vtop align
4222 \or
4223 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4224 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4225 \or
4226 \aftergroup#2% 14 \begingroup
4227 \else

```

```

4228      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4229      \fi
4230      \fi
4231      \bbl@dirlevel\currentgrouplevel
4232      \fi
4233      #1%
4234      \fi}
4235      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4236      \let\bbl@bodydir@gobble
4237      \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).

```

4238      \def\bbl@xebidipar{%
4239        \let\bbl@xebidipar\relax
4240        \TeXeTstate@ne
4241        \def\bbl@xeeverypar{%
4242          \ifcase\bbl@thepardir
4243            \ifcase\bbl@thetextdir\else\beginR\fi
4244          \else
4245            {\setbox\z@\lastbox\beginR\box\z@}%
4246          \fi}%
4247          \AddToHook{para/begin}{\bbl@xeeverypar}}
4248      \ifnum\bbl@bidimode>200 % Any xe bidi=
4249        \let\bbl@textdir@i@gobbletwo
4250        \let\bbl@xebidipar@empty
4251        \AddBabelHook{bidi}{foreign}{%
4252          \ifcase\bbl@thetextdir
4253            \BabelWrapText{\LR{##1}}%
4254          \else
4255            \BabelWrapText{\RL{##1}}%
4256          \fi}
4257        \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4258      \fi
4259      \fi

```

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

```

4260      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4261      \AtBeginDocument{%
4262        \ifx\pdfstringdefDisableCommands\undefined\else
4263          \ifx\pdfstringdefDisableCommands\relax\else
4264            \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4265          \fi
4266        \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`.

```

4267      \bbl@trace{Local Language Configuration}
4268      \ifx\loadlocalcfg\undefined
4269        \@ifpackagewith{babel}{noconfigs}%
4270        {\let\loadlocalcfg@gobble}%
4271        {\def\loadlocalcfg#1{%
4272          \InputIfFileExists{#1.cfg}%
4273          {\typeout{*****^J%
4274            * Local config file #1.cfg used^^J%
4275            *}}%

```

```

4276      \@empty}}
4277 \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).

```

4278 \bbl@trace{Language options}
4279 \def\BabelDefinitionFile#1#2#3{
4280 \let\bbl@afterlang\relax
4281 \let\BabelModifiers\relax
4282 \let\bbl@loaded\@empty
4283 \def\bbl@load@language#1{%
4284   \InputIfFileExists{#1.ldf}%
4285   {\edef\bbl@loaded{\CurrentOption
4286     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4287     \expandafter\let\expandafter\bbl@afterlang
4288       \csname\CurrentOption.ldf-h@k\endcsname
4289     \expandafter\let\expandafter\BabelModifiers
4290       \csname bbl@mod@\CurrentOption\endcsname
4291     \bbl@exp{\AtBeginDocument{%
4292       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4293     {\bbl@error{unknown-package-option}}{}}}%

```

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

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

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

```

4294 \ifx\GetDocumentProperties\undefined\else
4295   \let\bbl@beforeforeign\leavevmode
4296   \edef\bbl@tempa{\GetDocumentProperties{document/other-languages}}%
4297   \let\bbl@collect@other\@empty
4298   \bbl@foreach\bbl@tempa{%
4299     \edef\bbl@metalang{#1}%
4300     \ifx\bbl@metalang\@empty\else
4301       \begingroup
4302       \expandafter
4303       \bbl@bcplookup{\bbl@metalang}%-\@empty-\@empty-\@empty\@
4304       \ifx\bbl@bcp\relax
4305         \ifx\bbl@opt@main\@nnil
4306           \bbl@error{no-locale-for-meta}{\bbl@metalang}{}%
4307         \fi
4308       \else
4309         \bbl@read@ini{\bbl@bcp}\m@ne
4310         \xdef\bbl@collect@other{\bbl@collect@other,\language}%
4311         \GenericInfo{(babel) \spaces\spaces\spaces
4312           Passing '\language' to babel\@gobble}%
4313       \fi
4314     \endgroup
4315   \fi}
4316   \ifx\bbl@collect@other\@empty\else
4317     \xdef\bbl@language@opts{\bbl@collect@other,\bbl@language@opts}%
4318   \fi
4319   \edef\bbl@metalang{\GetDocumentProperties{document/language}}%
4320   \ifx\bbl@metalang\@empty\else
4321     \begingroup

```

```

4322 \expandafter
4323 \bbl@bcplookup{\bbl@metalang}%-\@empty-\@empty-\@empty\@@
4324 \ifx\bbl@bcp\relax
4325 \ifx\bbl@opt@main\@nnil
4326 \bbl@error{no-locale-for-meta}{\bbl@metalang}{}}}%
4327 \fi
4328 \else
4329 \bbl@read@ini{\bbl@bcp}\m@ne
4330 \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4331 \ifx\bbl@opt@main\@nnil
4332 \global\let\bbl@opt@main\language
4333 \fi
4334 \GenericInfo{}{(babel) \@spaces\@spaces\@spaces
4335 Passing '\language' to babel\@gobble}%
4336 \fi
4337 \endgroup
4338 \fi
4339 \fi
4340 \ifx\bbl@opt@config\@nnil
4341 \ifpackagewith{babel}{noconfigs}}}%
4342 {\InputIfFileExists{bblopts.cfg}%
4343 {\bbl@info{Configuration files are deprecated, as\\%
4344 they can break document portability.\\%
4345 Reported}%
4346 \typeout{*****^J%
4347 * Local config file bblopts.cfg used^J%
4348 *}}}%
4349 {}}}%
4350 \else
4351 \InputIfFileExists{\bbl@opt@config.cfg}%
4352 {\bbl@info{Configuration files are deprecated, as\\%
4353 they can break document portability.\\%
4354 Reported}%
4355 \typeout{*****^J%
4356 * Local config file \bbl@opt@config.cfg used^J%
4357 *}}}%
4358 {\bbl@error{config-not-found}}{}}}%
4359 \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`).

```

4360 \def\BabelBeforeIni#1#2{%
4361 \def\bbl@tempa{\@m}% <- Default if no \BDefFile
4362 \let\bbl@tempb\@empty
4363 #2%
4364 \edef\bbl@toload{%
4365 \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4366 \bbl@toload@last}%
4367 \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4368 \def\BabelDefinitionFile#1#2#3{%
4369 \def\bbl@tempa{#1}\def\bbl@tempb{#2}%
4370 \@namedef{\bbl@preldf@CurrentOption}{#3}%

```

```
4371 \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.

```
4372 \def\bbl@tempf{,}
4373 \@nameuse{clist_map_inline:Nn}\@raw@classoptionslist{%
4374   \in@{=}{#1}%
4375   \ifin@else
4376     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4377   \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.

```
4378 \let\bbl@toload\@empty
4379 \let\bbl@toload@last\@empty
4380 \let\bbl@unkopt\@gobble %<- Ugly
4381 \edef\bbl@tempc{%
4382   \bbl@tempf,@@,\bbl@language@opts
4383   \ifx\bbl@opt@main\@nnil\else,\bbl@opt@main\fi}
4384 \let\BabelLocalesTentative\bbl@tempc
4385 %
4386 \bbl@foreach\bbl@tempc{%
4387   \in@{@@}{#1}%<- Ugly
4388   \ifin@
4389     \def\bbl@unkopt##1{%
4390       \DeclareOption{##1}{\bbl@error{unknown-package-option}{}}}%
4391   \else
4392     \def\CurrentOption{#1}%
4393     \bbl@xin@{//#1//}{\bbl@toload@last}% Collapse consecutive
4394     \ifin@else
4395       \lowercase{\InputIfFileExists{babel-#1.tex}}{%
4396         \IfFileExists{#1.ldf}%
4397           {\edef\bbl@toload{%
4398             \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4399             \bbl@toload@last}%
4400             \edef\bbl@toload@last{0/0//\CurrentOption//und/#1}}%
4401             {\bbl@unkopt{#1}}}%
4402     \fi
4403   \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.

```
4404 \ifx\bbl@opt@main\@nnil
4405   \ifx\bbl@toload@last\@empty
4406     \def\bbl@toload@last{0/0//nil//und-x-nil/nil}
4407     \bbl@info{%
4408       You haven't specified a language as a class or package\%
4409       option. I'll load 'nil'. Reported}
4410   \fi
4411 \else
4412   \let\bbl@tempc\@empty
4413   \bbl@foreach\bbl@toload{%
4414     \bbl@xin@{//#1//}{\bbl@opt@main//}{#1}%
4415     \ifin@else
4416       \bbl@add@list\bbl@tempc{#1}%
4417     \fi}
4418   \let\bbl@toload\bbl@tempc
4419 \fi
4420 \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’.

```

4421 \def\AfterBabelLanguage#1{%
4422   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4423 \NewHook{babel/presets}
4424 \UseHook{babel/presets}
4425 %
4426 \let\bbl@tempb\@empty
4427 \def\bbl@tempc#1/#2//#3//#4/#5\@{%
4428   \count@\z@
4429   \ifnum#2=\@m % if no \BabelDefinitionFile
4430     \ifnum#1=\z@ % not main. -- % if provide+=!, provide*=!
4431       \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0//#3//#4/#3\@
4432       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4433       \fi
4434     \else % 10 = main -- % if provide+=!, provide*=!
4435       \ifodd\bbl@ldfflag\bbl@tempc 10/0//#3//#4/#3\@
4436       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4437       \fi
4438     \fi
4439   \else
4440     \ifnum#1=\z@ % not main
4441       \ifnum\bbl@iniflag>\@ne\else % if 0, provide
4442         \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4443       \fi
4444     \else % 10 = main
4445       \ifodd\bbl@iniflag\else % if provide+, provide*
4446         \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4447       \fi
4448     \fi
4449     \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4450   \fi}

```

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

```

4451 \def\bbl@tempd#1#2#3#4#5{%
4452   \DeclareOption{#3}{%
4453     \ifcase\count@
4454       \bbl@exp{\bbl@add\bbl@tempb{%
4455         \global\let\bbl@afterload\@empty
4456         \nameuse\bbl@preini{#3}%
4457         \bbl@ldfinit
4458         \def\CurrentOption{#3}%
4459         \babelprovide[import=#4,\ifnum#1=\z@\else\bbl@opt@provide,main\fi]{#3}%
4460         \bbl@afterldf
4461         \bbl@afterload}}%
4462     \else
4463       \bbl@add\bbl@tempb{%
4464         \global\let\bbl@afterload\@empty
4465         \def\CurrentOption{#3}%
4466         \let\localename\CurrentOption
4467         \let\languagename\localename
4468         \def\BabelIniTag{#4}%
4469         \nameuse\bbl@preldf{#3}%
4470         \begingroup
4471           \bbl@id@assign
4472           \bbl@read@ini{\BabelIniTag}0%
4473         \endgroup
4474         \bbl@load@language{#5}%
4475         \bbl@afterload}%
4476     \fi}

```

```

4477 %
4478 \bbl@foreach\bbl@toload{\bbl@tempc#1\@@}
4479 \bbl@tempb
4480 \DeclareOption*{}
4481 \ProcessOptions
4482 %
4483 \bbl@exp{%
4484   \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{\begindocument}{}}}%
4485 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}}{}
4486 \AddToHook{begindocument/end}{%
4487   \bbl@info{Main language set to '\mainlocalename'\@gobble}}
4488 </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`

```

4489 <*kernel>
4490 \let\bbl@onlyswitch\@empty
4491 \input babel.def
4492 \let\bbl@onlyswitch\@undefined
4493 </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.

```

4494 <*errors>
4495 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4496 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4497 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4498 \catcode`\@=11 \catcode`\^=7
4499 %
4500 \ifx\MessageBreak\@undefined
4501   \gdef\bbl@error@i#1#2{%
4502     \begingroup
4503       \newlinechar=`^^J
4504       \def\{^^J(babel) }%
4505       \errhelp{#2}\errmessage{\{#1}%
4506     \endgroup}
4507 \else
4508   \gdef\bbl@error@i#1#2{%
4509     \begingroup
4510       \def\{^^J\MessageBreak}%
4511       \PackageError{babel}{#1}{#2}%
4512     \endgroup}
4513 \fi
4514 \def\bbl@errmessage#1#2#3{%
4515   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4516     \bbl@error@i{#2}{#3}}

```

```

4517% Implicit #2#3#4:
4518\gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4519%
4520\bbl@errmessage{not-yet-available}
4521    {Not yet available}%
4522    {Find an armchair, sit down and wait}
4523\bbl@errmessage{bad-package-option}%
4524    {Bad option '#1=#2'. Either you have misspelled the\\%
4525    key or there is a previous setting of '#1'. Valid\\%
4526    keys are, among others, 'shorthands', 'main', 'bidi',\\%
4527    'strings', 'config', 'headfoot', 'safe', 'math'.}%
4528    {See the manual for further details.}
4529\bbl@errmessage{base-on-the-fly}
4530    {For a language to be defined on the fly 'base'\\%
4531    is not enough, and the whole package must be\\%
4532    loaded. Either delete the 'base' option or\\%
4533    request the languages explicitly}%
4534    {See the manual for further details.}
4535\bbl@errmessage{undefined-language}
4536    {You haven't defined the language '#1' yet.\\%
4537    Perhaps you misspelled it or your installation\\%
4538    is not complete}%
4539    {Your command will be ignored, type <return> to proceed}
4540\bbl@errmessage{invalid-ini-name}
4541    {'#1' not valid with the 'ini' mechanism.\\%
4542    I think you want '#2' instead. You may continue,\\%
4543    but you should fix the name. See the babel manual\\%
4544    for the available locales with 'provide'}%
4545    {See the manual for further details.}
4546\bbl@errmessage{shorthand-is-off}
4547    {I can't declare a shorthand turned off (\string#2)}
4548    {Sorry, but you can't use shorthands which have been\\%
4549    turned off in the package options}
4550\bbl@errmessage{not-a-shorthand}
4551    {The character '\string #1' should be made a shorthand character;\\%
4552    add the command \string\usesshorthands\string{#1\string} to
4553    the preamble.\\%
4554    I will ignore your instruction}%
4555    {You may proceed, but expect unexpected results}
4556\bbl@errmessage{not-a-shorthand-b}
4557    {I can't switch '\string#2' on or off--not a shorthand\\%
4558    This character is not a shorthand. Maybe you made\\%
4559    a typing mistake?}%
4560    {I will ignore your instruction.}
4561\bbl@errmessage{unknown-attribute}
4562    {The attribute #2 is unknown for language #1.}%
4563    {Your command will be ignored, type <return> to proceed}
4564\bbl@errmessage{missing-group}
4565    {Missing group for string \string#1}%
4566    {You must assign strings to some category, typically\\%
4567    captions or extras, but you set none}
4568\bbl@errmessage{only-lua-xe}
4569    {This macro is available only in LuaLaTeX and XeLaTeX.}%
4570    {Consider switching to these engines.}
4571\bbl@errmessage{only-lua}
4572    {This macro is available only in LuaLaTeX}%
4573    {Consider switching to that engine.}
4574\bbl@errmessage{unknown-provide-key}
4575    {Unknown key '#1' in \string\babelprovide}%
4576    {See the manual for valid keys}%
4577\bbl@errmessage{unknown-mapfont}
4578    {Option '\bbl@KVP@mapfont' unknown for\\%
4579    mapfont. Use 'direction'}%

```

```

4580 {See the manual for details.}
4581 \bbl@errmessage{no-ini-file}
4582 {There is no ini file for the requested language\\%
4583 (#1: \language). Perhaps you misspelled it or your\\%
4584 installation is not complete}%
4585 {Fix the name or reinstall babel.}
4586 \bbl@errmessage{digits-is-reserved}
4587 {The counter name 'digits' is reserved for mapping\\%
4588 decimal digits}%
4589 {Use another name.}
4590 \bbl@errmessage{limit-two-digits}
4591 {Currently two-digit years are restricted to the\\
4592 range 0-9999}%
4593 {There is little you can do. Sorry.}
4594 \bbl@errmessage{alphabetic-too-large}
4595 {Alphabetic numeral too large (#1)}%
4596 {Currently this is the limit.}
4597 \bbl@errmessage{no-ini-info}
4598 {I've found no info for the current locale.\\%
4599 The corresponding ini file has not been loaded\\%
4600 Perhaps it doesn't exist}%
4601 {See the manual for details.}
4602 \bbl@errmessage{unknown-ini-field}
4603 {Unknown field '#1' in \string\BCPdata.\\%
4604 Perhaps you misspelled it}%
4605 {See the manual for details.}
4606 \bbl@errmessage{unknown-locale-key}
4607 {Unknown key for locale '#2':\\%
4608 #3\\%
4609 \string#1 will be set to \string\relax}%
4610 {Perhaps you misspelled it.}%
4611 \bbl@errmessage{adjust-only-vertical}
4612 {Currently, #1 related features can be adjusted only\\%
4613 in the main vertical list}%
4614 {Maybe things change in the future, but this is what it is.}
4615 \bbl@errmessage{layout-only-vertical}
4616 {Currently, layout related features can be adjusted only\\%
4617 in vertical mode}%
4618 {Maybe things change in the future, but this is what it is.}
4619 \bbl@errmessage{bidi-only-lua}
4620 {The bidi method 'basic' is available only in\\%
4621 luatex. I'll continue with 'bidi=default', so\\%
4622 expect wrong results.\\%
4623 Suggested actions:\\%
4624 * If possible, switch to luatex, as xetex is not\\%
4625 recommend anymore.\\
4626 * If you can't, try 'bidi=bidi' with xetex.\\%
4627 * With pdftex, only 'bidi=default' is available.}%
4628 {See the manual for further details.}
4629 \bbl@errmessage{multiple-bidi}
4630 {Multiple bidi settings inside a group\\%
4631 I'll insert a new group, but expect wrong results.\\%
4632 Suggested action:\\%
4633 * Add a new group where appropriate.}
4634 {See the manual for further details.}
4635 \bbl@errmessage{unknown-package-option}
4636 {Unknown option '\CurrentOption'.\\%
4637 Suggested actions:\\%
4638 * Make sure you haven't misspelled it\\%
4639 * Check in the babel manual that it's supported\\%
4640 * If supported and it's a language, you may\\%
4641 \space\space need in some distributions a separate\\%
4642 \space\space installation\\%

```

```

4643 * If installed, check there isn't an old\\%
4644 \space\space version of the required files in your system\\%
4645 * If it's an unsupported language, create it with\\%
4646 \string\babelprovide (see the manual)}
4647 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4648 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4649 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4650 \bbl@errmessage{config-not-found}
4651 {Local config file '\bbl@opt@config.cfg' not found.\\%
4652 Suggested actions:\\%
4653 * Make sure you haven't misspelled it in config=\\%
4654 * Check it exists and it's in the correct path}%
4655 {Perhaps you misspelled it.}
4656 \bbl@errmessage{late-after-babel}
4657 {Too late for \string\AfterBabelLanguage}%
4658 {Languages have been loaded, so I can do nothing}
4659 \bbl@errmessage{double-hyphens-class}
4660 {Double hyphens aren't allowed in \string\babelcharclass\\%
4661 because it's potentially ambiguous}%
4662 {See the manual for further info}
4663 \bbl@errmessage{unknown-interchar}
4664 {'#1' for '\language' cannot be enabled.\\%
4665 Maybe there is a typo}%
4666 {See the manual for further details.}
4667 \bbl@errmessage{unknown-interchar-b}
4668 {'#1' for '\language' cannot be disabled.\\%
4669 Maybe there is a typo}%
4670 {See the manual for further details.}
4671 \bbl@errmessage{charproperty-only-vertical}
4672 {\string\babelcharproperty\space can be used only in\\%
4673 vertical mode (preamble or between paragraphs)}%
4674 {See the manual for further info}
4675 \bbl@errmessage{unknown-char-property}
4676 {No property named '#2'. Allowed values are\\%
4677 direction (bc), mirror (bmg), and linebreak (lb)}%
4678 {See the manual for further info}
4679 \bbl@errmessage{bad-transform-option}
4680 {Bad option '#1' in a transform.\\%
4681 I'll ignore it but expect more errors}%
4682 {See the manual for further info.}
4683 \bbl@errmessage{font-conflict-transforms}
4684 {Transforms cannot be re-assigned to different\\%
4685 fonts. The conflict is in '\bbl@kv@label'.\\%
4686 Apply the same fonts or use a different label}%
4687 {See the manual for further details.}
4688 \bbl@errmessage{transform-not-available}
4689 {'#1' for '\language' cannot be enabled.\\%
4690 Maybe there is a typo or it's a font-dependent transform}%
4691 {See the manual for further details.}
4692 \bbl@errmessage{transform-not-available-b}
4693 {'#1' for '\language' cannot be disabled.\\%
4694 Maybe there is a typo or it's a font-dependent transform}%
4695 {See the manual for further details.}
4696 \bbl@errmessage{year-out-range}
4697 {Year out of range.\\%
4698 The allowed range is #1}%
4699 {See the manual for further details.}
4700 \bbl@errmessage{only-pdf-tex-lang}
4701 {The '#1' ldf style doesn't work with #2,\\%
4702 but you can use the ini locale instead.\\%
4703 Try adding 'provide=' to the option list. You may\\%
4704 also want to set 'bidi=' to some value}%
4705 {See the manual for further details.}

```

```

4706 \bbl@errmessage{hyphenmins-args}
4707 {\string\babelhyphenmins\ accepts either the optional\\%
4708 argument or the star, but not both at the same time}%
4709 {See the manual for further details.}
4710 \bbl@errmessage{no-locale-for-meta}
4711 {There isn't currently a locale for the 'lang' requested\\%
4712 in the PDF metadata ('#1'). To fix it, you can\\%
4713 set explicitly a similar language (using the same\\%
4714 script) with the key main= when loading babel. If you\\%
4715 continue, I'll fallback to the 'nil' language, with\\%
4716 tag 'und' and script 'Latn', but expect a bad font\\%
4717 rendering with other scripts. You may also need set\\%
4718 explicitly captions and date, too}%
4719 {See the manual for further details.}
4720 </errors>
4721 <*patterns>

```

8. Loading hyphenation patterns

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

```

4722 <@Make sure ProvidesFile is defined@>
4723 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4724 \xdef\bbl@format{\jobname}
4725 \def\bbl@version{<@version@>}
4726 \def\bbl@date{<@date@>}
4727 \ifx\AtBeginDocument\undefined
4728 \def\@empty{}
4729 \fi
4730 <@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.

```

4731 \def\process@line#1#2 #3 #4 {%
4732 \ifx=#1%
4733 \process@synonym{#2}%
4734 \else
4735 \process@language{#1#2}{#3}{#4}%
4736 \fi
4737 \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.

```

4738 \toks@{}
4739 \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.

```

4740 \def\process@synonym#1{%
4741 \ifnum\last@language=\m@ne
4742 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4743 \else
4744 \expandafter\chardef\csname l@#1\endcsname\last@language
4745 \wlog{\string\l@#1=\string\language\the\last@language}%
4746 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4747 \csname\language\hyphenmins\endcsname

```

```

4748 \let\bbl@elt\relax
4749 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}{}{}%
4750 \fi}

```

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

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

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

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

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

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

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

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

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

```

4751 \def\process@language#1#2#3{%
4752 \expandafter\addlanguage\csname l@#1\endcsname
4753 \expandafter\language\csname l@#1\endcsname
4754 \edef\languagename{#1}%
4755 \bbl@hook@everylanguage{#1}%
4756 % > luatex
4757 \bbl@get@enc#1::\@@@
4758 \begingroup
4759 \lefthyphenmin\m@ne
4760 \bbl@hook@loadpatterns{#2}%
4761 % > luatex
4762 \ifnum\lefthyphenmin=\m@ne
4763 \else
4764 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4765 \the\lefthyphenmin\the\righthyphenmin}%
4766 \fi
4767 \endgroup
4768 \def\bbl@tempa{#3}%
4769 \ifx\bbl@tempa\@empty\else
4770 \bbl@hook@loadexceptions{#3}%
4771 % > luatex
4772 \fi
4773 \let\bbl@elt\relax
4774 \edef\bbl@languages{%
4775 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4776 \ifnum\the\language=\z@
4777 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4778 \set@hyphenmins\thr@@\relax
4779 \else
4780 \expandafter\expandafter\expandafter\set@hyphenmins
4781 \csname #1hyphenmins\endcsname

```

```

4782 \fi
4783 \the\toks@
4784 \toks@{}%
4785 \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.

```

4786 \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.

```

4787 \def\bbl@hook@everylanguage#1{}
4788 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4789 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4790 \def\bbl@hook@loadkernel#1{%
4791   \def\addlanguage{\csname newlanguage\endcsname}%
4792   \def\adddialect##1##2{%
4793     \global\chardef##1##2\relax
4794     \wlog{string##1 = a dialect from \string\language##2}}%
4795   \def\iflanguage##1{%
4796     \expandafter\ifx\csname l@##1\endcsname\relax
4797       \nolannerr{##1}%
4798     \else
4799       \ifnum\csname l@##1\endcsname=\language
4800         \expandafter\expandafter\expandafter\@firstoftwo
4801       \else
4802         \expandafter\expandafter\expandafter\@secondoftwo
4803       \fi
4804     \fi}%
4805   \def\providehyphenmins##1##2{%
4806     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4807       \namedef{##1hyphenmins}{##2}%
4808     \fi}%
4809   \def\set@hyphenmins##1##2{%
4810     \lefthyphenmin##1\relax
4811     \righthyphenmin##2\relax}%
4812   \def\selectlanguage{%
4813     \errhelp{Selecting a language requires a package supporting it}%
4814     \errmessage{No multilingual package has been loaded}}%
4815   \let\foreignlanguage\selectlanguage
4816   \let\otherlanguage\selectlanguage
4817   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4818   \def\bbl@usehooks##1##2{%
4819     \def\setlocale{%
4820       \errhelp{Find an armchair, sit down and wait}%
4821       \errmessage{(babel) Not yet available}}%
4822     \let\uselocale\setlocale
4823     \let\locale\setlocale
4824     \let\selectlocale\setlocale
4825     \let\localename\setlocale
4826     \let\textlocale\setlocale
4827     \let\textlanguage\setlocale
4828     \let\languagetext\setlocale}
4829   \begingroup
4830     \def\AddBabelHook#1#2{%
4831       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4832         \def\next{\toks1}%
4833       \else
4834         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4835       \fi
4836     \next}

```

```

4837 \ifx\directlua\@undefined
4838 \ifx\XeTeXinputencoding\@undefined\else
4839 \input xebabel.def
4840 \fi
4841 \else
4842 \input luababel.def
4843 \fi
4844 \openin1 = babel-\bbl@format.cfg
4845 \ifeof1
4846 \else
4847 \input babel-\bbl@format.cfg\relax
4848 \fi
4849 \closein1
4850 \endgroup
4851 \bbl@hook@loadkernel{switch.def}

```

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

```

4852 \openin1 = language.dat

```

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

```

4853 \def\language{english}%
4854 \ifeof1
4855 \message{I couldn't find the file language.dat,\space
4856         I will try the file hyphen.tex}
4857 \input hyphen.tex\relax
4858 \chardef\l@english\z@
4859 \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`.

```

4860 \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.

```

4861 \loop
4862 \endlinechar@m@ne
4863 \read1 to \bbl@line
4864 \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.

```

4865 \if T\ifeof1F\fi T\relax
4866 \ifx\bbl@line\empty\else
4867 \edef\bbl@line{\bbl@line\space\space\space}%
4868 \expandafter\process@line\bbl@line\relax
4869 \fi
4870 \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.

```

4871 \begingroup
4872 \def\bbl@elt#1#2#3#4{%
4873 \global\language=#2\relax
4874 \gdef\language{#1}%
4875 \def\bbl@elt##1##2##3##4{}}%
4876 \bbl@languages
4877 \endgroup
4878 \fi
4879 \closein1

```

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

```
4880 \if/\the\toks@\else
4881 \errhelp{language.dat loads no language, only synonyms}
4882 \errmessage{Orphan language synonym}
4883 \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.

```
4884 \let\bbl@line\@undefined
4885 \let\process@line\@undefined
4886 \let\process@synonym\@undefined
4887 \let\process@language\@undefined
4888 \let\bbl@get@enc\@undefined
4889 \let\bbl@hyph@enc\@undefined
4890 \let\bbl@tempa\@undefined
4891 \let\bbl@hook@loadkernel\@undefined
4892 \let\bbl@hook@everylanguage\@undefined
4893 \let\bbl@hook@loadpatterns\@undefined
4894 \let\bbl@hook@loadexceptions\@undefined
4895 </patterns>
```

Here the code for `initTeX` 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`).

```
4896 <<*More package options>> ≡
4897 \chardef\bbl@bidimode\z@
4898 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4899 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4900 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4901 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4902 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4903 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4904 <</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`.

```
4905 <<*Font selection>> ≡
4906 \bbl@trace{Font handling with fontspec}
4907 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4908 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4909 \DisableBabelHook{babel-fontspec}
4910 \@onlypreamble\babelfont
4911 \ifx\NewDocumentCommand\@undefined\else % Not plain
4912 \NewDocumentCommand\babelfont{0}{m0}{m0}{}%
4913 \bbl@bblfont@o[#1]{#2}[#3,#5]{#4}}
4914 \fi
4915 \newcommand\bbl@bblfont@o[2][]{% 1=langs/scripts 2=fam
4916 \ifx\fontspec\@undefined
4917 \usepackage{fontspec}%
4918 \fi
4919 \EnableBabelHook{babel-fontspec}%
4920 \edef\bbl@tempa{#1}%
4921 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4922 \bbl@bblfont}
4923 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
```

```

4924 \bbl@ifunset{\bbl@tempb family}%
4925   {\bbl@providfam{\bbl@tempb}}%
4926   {}%
4927 % For the default font, just in case:
4928 \bbl@ifunset{\bbl@lsys{\language}\bbl@provide@lsys{\language}}{%}%
4929 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4930   {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
4931   \bbl@exp{%
4932     \let<\bbl@tempb dflt@\language><\bbl@tempb dflt@>%
4933     \\bbl@font@set<\bbl@tempb dflt@\language>%
4934     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4935   {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *srt
4936     \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:

```

4937 \def\bbl@providfam#1{%
4938   \bbl@exp{%
4939     \\newcommand<#1default>{}% Just define it
4940     \\bbl@add@list\\bbl@font@fams{#1}%
4941     \\NewHook{#1family}%
4942     \\DeclareRobustCommand<#1family>{%
4943       \\not@math@alphabet<#1family>\relax
4944       % \\prepare@family@series@update{#1}<#1default>% TODO. Fails
4945       \\fontfamily<#1default>%
4946       \\UseHook{#1family}%
4947       \\selectfont}%
4948     \\DeclareTextFontCommand{\<text#1>}{<#1family>}}

```

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

```

4949 \def\bbl@nostdfont#1{%
4950   \bbl@once{nostdfam-\f@family}%
4951   {\bbl@infowarn{The current font is not a babel standard family:\\%
4952     #1%
4953     \fontname\font\\%
4954     There is nothing intrinsically wrong, and you can\\%,
4955     ignore this message altogether if you do not need\\%
4956     this font. If they are used in the document, be aware\\%
4957     'babel' will not set Script and Language for it, so\\%
4958     you may consider defining a new family with \string\babelfont.\\%
4959     See the manual for further details about \string\babelfont.
4960     Reported}}%
4961   }%
4962 \gdef\bbl@switchfont{%
4963   \bbl@ifunset{\bbl@lsys{\language}\bbl@provide@lsys{\language}}{%}%
4964   \bbl@exp{% e.g., Arabic -> arabic
4965     \lowercase{\edef\\bbl@tempa{\bbl@c{l}{sname}}}%
4966     \bbl@foreach\bbl@font@fams{%
4967       \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4968       {\bbl@ifunset{\bbl@##1dflt*~\bbl@tempa}% (2) from script?
4969         {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4970           {}% 123=F - nothing!
4971           {\bbl@exp{% 3=T - from generic
4972             \global\let<\bbl@##1dflt@\language>%
4973             \<\bbl@##1dflt@>}}}%
4974           {\bbl@exp{% 2=T - from script
4975             \global\let<\bbl@##1dflt@\language>%
4976             \<\bbl@##1dflt*~\bbl@tempa>}}}%
4977           {}% 1=T - language, already defined
4978     \def\bbl@tempa{\bbl@nostdfont}}}%
4979   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4980     \bbl@ifunset{\bbl@##1dflt@\language}%
4981     {\bbl@cs{famrst@##1}%
4982     \global\bbl@csarg\let{famrst@##1}\relax}%

```

```

4983      {\bbl@exp{% order is relevant.
4984          \\bbl@add\\originalTeX{%
4985              \\bbl@font@rst{\bbl@cfl{##1dflt}}}%
4986                  \<##1default>\<##1family>{##1}}%
4987          \\bbl@font@set{\bbl@##1dflt@\languagefamily}% the main part!
4988              \<##1default>\<##1family>}}}%
4989      \bbl@ifrestoring{\bbl@tempa}}%

```

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

```

4990 \ifx\f@family\undefined\else      % if latex
4991     \ifcase\bbl@engine                % if pdftex
4992         \let\bbl@cckcstdfonts\relax
4993     \else
4994         \def\bbl@cckcstdfonts{%
4995             \begingroup
4996             \global\let\bbl@cckcstdfonts\relax
4997             \let\bbl@tempa\empty
4998             \bbl@foreach\bbl@font@fams{%
4999                 \bbl@ifunset{\bbl@##1dflt@}%
5000                 {\@nameuse{##1family}}%
5001                 \bbl@csarg\gdef{WFF@\f@family}}}% Flag
5002                 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
5003                     \space\space\fontname\font\\}%
5004                 \bbl@csarg\xdef{##1dflt@}{\f@family}}%
5005                 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
5006                 {}}%
5007     \ifx\bbl@tempa\empty\else
5008         \bbl@infowarn{The following font families will use the default\\%
5009             settings for all or some languages:\\%
5010             \bbl@tempa
5011             There is nothing intrinsically wrong with it, but\\%
5012             'babel' will no set Script and Language, which could\\%
5013             be relevant in some languages. If your document uses\\%
5014             these families, consider redefining them with \string\babelfont.\\%
5015             Reported}%
5016     \fi
5017 \endgroup
5018 \fi
5019 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

5020 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
5021     \bbl@xin@{<>}{#1}%
5022     \ifin@
5023         \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
5024     \fi
5025     \bbl@exp{%
5026         \def\\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
5027         \\bbl@ifsamestring{#2}{\f@family}%
5028         {\#3
5029             \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
5030     \let\\bbl@tempa\relax}%
5031     {}%

```

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).

```

5032 \def\bbbl@fontspec@set#1#2#3#4{% eg \bbbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
5033 \let\bbbl@tempe\bbbl@mapselect
5034 \edef\bbbl@tempb{\bbbl@stripslash#4}% Catcodes hack (better pass it).
5035 \bbbl@exp{\bbbl@replace\bbbl@tempb{\bbbl@stripslash\family/}}}%
5036 \let\bbbl@mapselect\relax
5037 \let\bbbl@temp@fam#4% e.g., '\rmfamily', to be restored below
5038 \let#4@empty % Make sure \renewfontfamily is valid
5039 \bbbl@set@renderer
5040 \bbbl@exp{%
5041 \let\bbbl@temp@pfam<\bbbl@stripslash#4\space> e.g., '\rmfamily '
5042 \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbbl@cl{sname}}}%
5043 {\bbbl@newfontscript{\bbbl@cl{sname}}{\bbbl@cl{sotf}}}%
5044 \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbbl@cl{lname}}}%
5045 {\bbbl@newfontlanguage{\bbbl@cl{lname}}{\bbbl@cl{lotf}}}%
5046 \bbbl@renewfontfamily\#4%
5047 [\bbbl@cl{lsys},% xetex removes unknown features :-(
5048 \ifcase\bbbl@engine\or RawFeature={family=\bbbl@tempb},\fi
5049 #2]}{#3}% i.e., \bbbl@exp{..}{#3}
5050 \bbbl@unset@renderer
5051 \begingroup
5052 #4%
5053 \xdef#1{\f@family}% e.g., \bbbl@rmdflt@lang{FreeSerif(0)}
5054 \endgroup
5055 \bbbl@xin@{\string>\string s\string s\string u\string b\string*}%
5056 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
5057 \ifin@
5058 \global\bbbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
5059 \fi
5060 \bbbl@xin@{\string>\string s\string s\string u\string b\string*}%
5061 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
5062 \ifin@
5063 \global\bbbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
5064 \fi
5065 \let#4\bbbl@temp@fam
5066 \bbbl@exp{\let<\bbbl@stripslash#4\space>\bbbl@temp@pfam
5067 \let\bbbl@mapselect\bbbl@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.

```

5068 \def\bbbl@font@rst#1#2#3#4{%
5069 \bbbl@ccarg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}

```

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

```

5070 \def\bbbl@font@fams{rm,sf,tt}
5071 <</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.

```

5072 <{*xetex}>
5073 \def\BabelStringsDefault{unicode}
5074 \let\xebbl@stop\relax

```

```

5075 \AddBabelHook{xetex}{encodedcommands}{%
5076   \def\bbl@tempa{#1}%
5077   \ifx\bbl@tempa\@empty
5078     \XeTeXinputencoding"bytes"%
5079   \else
5080     \XeTeXinputencoding"#1"%
5081   \fi
5082   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
5083 \AddBabelHook{xetex}{stopcommands}{%
5084   \xebbl@stop
5085   \let\xebbl@stop\relax}
5086 \def\bbl@input@classes{% Used in CJK intraspaces
5087   \input{load-unicode-xetex-classes.tex}%
5088   \let\bbl@input@classes\relax}
5089 \def\bbl@intraspace#1 #2 #3\@@{%
5090   \bbl@csarg\gdef{xisp@\language}%
5091   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
5092 \def\bbl@intrapenalty#1\@@{%
5093   \bbl@csarg\gdef{xipn@\language}%
5094   {\XeTeXlinebreakpenalty #1\relax}}
5095 \def\bbl@provide@intraspace{%
5096   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
5097   \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{lnbrk}} \fi
5098   \ifin@
5099     \bbl@ifunset{bbl@intsp@\language}{}%
5100     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
5101       \ifx\bbl@KVP@intraspace\@nnil
5102         \bbl@exp{%
5103           \\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5104         \fi
5105         \ifx\bbl@KVP@intrapenalty\@nnil
5106           \bbl@intrapenalty0\@@
5107         \fi
5108         \fi
5109         \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
5110           \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
5111         \fi
5112         \ifx\bbl@KVP@intrapenalty\@nnil\else
5113           \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5114         \fi
5115         \bbl@exp{%
5116           \\bbl@add\<extras\language>{%
5117             \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
5118             \<bbl@xisp@\language>%
5119             \<bbl@xipn@\language>}%
5120           \\bbl@tglobal\<extras\language>%
5121           \\bbl@add\<noextras\language>{%
5122             \XeTeXlinebreaklocale ""}%
5123           \\bbl@tglobal\<noextras\language>}%
5124           \ifx\bbl@ispacesize\@undefined
5125             \gdef\bbl@ispacesize{\bbl@cl{xisp}}}%
5126           \ifx\AtBeginDocument\@notprerr
5127             \expandafter\@secondoftwo % to execute right now
5128           \fi
5129           \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
5130         \fi}%
5131   \fi}
5132 \ifx\DisableBabelHook\@undefined\endinput\fi
5133 \let\bbl@set@renderer\relax
5134 \let\bbl@unset@renderer\relax
5135 <@Font selection@>
5136 \def\bbl@provide@extra#1{}

```

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

```

5137 \def\bbl@xenoxyph@d{%
5138   \bbl@ifset{bbl@prehc\language}%
5139     {\ifnum\hyphenchar\font=\defaultshphenchar
5140       \iffontchar\font\bbl@cl{prehc}\relax
5141       \hyphenchar\font\bbl@cl{prehc}\relax
5142     \else\iffontchar\font"200B
5143       \hyphenchar\font"200B
5144     \else
5145       \bbl@warning
5146         {Neither 0 nor ZERO WIDTH SPACE are available\\%
5147          in the current font, and therefore the hyphen\\%
5148          will be printed. Try changing the fontspec's\\%
5149          'HyphenChar' to another value, but be aware\\%
5150          this setting is not safe (see the manual).\\%
5151          Reported}%
5152       \hyphenchar\font\defaultshphenchar
5153     \fi\fi
5154   \fi}%
5155   {\hyphenchar\font\defaultshphenchar}}

```

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.

```

5156 \ifnum\Xe@alloc@intercharclass<\thr@@
5157   \Xe@alloc@intercharclass\thr@@
5158 \fi
5159 \chardef\bbl@xeclasse@default@=\z@
5160 \chardef\bbl@xeclasse@cjkkideogram@=\@ne
5161 \chardef\bbl@xeclasse@cjklleftpunctuation@=\tw@
5162 \chardef\bbl@xeclasse@cjkrighpunctuation@=\thr@@
5163 \chardef\bbl@xeclasse@boundary@=4095
5164 \chardef\bbl@xeclasse@ignore@=4096

```

The machinery is activated with a hook (enabled only if actually used). Here `\bbl@tempc` is pre-set with `\bbl@usingxeclasse`, 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.

```

5165 \AddBabelHook{babel-interchar}{beforeextras}{%
5166   \@nameuse{bbl@xechars\language}}
5167 \DisableBabelHook{babel-interchar}
5168 \protected\def\bbl@charclass#1{%
5169   \ifnum\count@<\z@
5170     \count@-\count@
5171   \loop
5172     \bbl@exp{%
5173       \\\babel@savevariable{\XeTeXcharclass`Uchar\count@}}%
5174     \XeTeXcharclass\count@ \bbl@tempc
5175     \ifnum\count@<`#1\relax
5176     \advance\count@ \@ne
5177   \repeat
5178 \else
5179   \babel@savevariable{\XeTeXcharclass`#1}%
5180   \XeTeXcharclass`#1 \bbl@tempc
5181 \fi
5182 \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@usingxeclasse\bbl@xeclasse@punct@english\bbl@charclass{.}` `\bbl@charclass{,}` (etc.), where `\bbl@usingxeclasse` 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.

```

5183 \newcommand\bbl@ifinterchar[1]{%
5184   \let\bbl@tempa\@gobble      % Assume to ignore
5185   \edef\bbl@tempb{\zap@space#1 \@empty}%
5186   \ifx\bbl@KVP@interchar\@nnil\else
5187     \bbl@replace\bbl@KVP@interchar{ },{,%
5188     \bbl@foreach\bbl@tempb{%
5189       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,%
5190       \ifin@
5191         \let\bbl@tempa\@firstofone
5192       \fi}%
5193   \fi
5194   \bbl@tempa}
5195 \newcommand\IfBabelIntercharT[2]{%
5196   \bbl@carg\bbl@add{\bbl@icsave\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5197 \newcommand\babelcharclass[3]{%
5198   \EnableBabelHook{babel-interchar}%
5199   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5200   \def\bbl@tempb##1{%
5201     \ifx##1\@empty\else
5202       \ifx##1-%
5203         \bbl@upto
5204       \else
5205         \bbl@charclass{%
5206           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5207       \fi
5208       \expandafter\bbl@tempb
5209     \fi}%
5210   \bbl@ifunset{\bbl@xechars@#1}%
5211   {\toks@{%
5212     \babel@savevariable\XeTeXinterchartokenstate
5213     \XeTeXinterchartokenstate\@ne
5214   }}%
5215   {\toks@\expandafter\expandafter\expandafter{%
5216     \csname bbl@xechars@#1\endcsname}}%
5217   \bbl@csarg\edef{xechars@#1}{%
5218     \the\toks@
5219     \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5220     \bbl@tempb#3\@empty}}
5221 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
5222 \protected\def\bbl@upto{%
5223   \ifnum\count@>\z@
5224     \advance\count@\@ne
5225     \count@-\count@
5226   \else\ifnum\count@=\z@
5227     \bbl@charclass{-}%
5228   \else
5229     \bbl@error{double-hyphens-class}{ }{}{}%
5230   \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 an intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<language>`.

```

5231 \def\bbl@ignoreinterchar{%
5232   \ifnum\language=\l@nohyphenation
5233     \expandafter\@gobble
5234   \else
5235     \expandafter\@firstofone
5236   \fi}
5237 \newcommand\babelinterchar[5][]{%
5238   \let\bbl@kv@label\@empty
5239   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%

```

```

5240 \namedef{\zap@space bbl@xeinter@bbl@kv@label @#3@#4@#2 \@empty}%
5241 {\bbl@ignoreinterchar{#5}}%
5242 \bbl@csarg\let{ic@bbl@kv@label @#2}\@firstofone
5243 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5244 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5245 \XeTeXinterchartoks
5246 \@nameuse{bbl@xeclasse@bbl@tempa @#2}%
5247 \bbl@ifunset{bbl@xeclasse@bbl@tempa @#2}{#2}} %
5248 \@nameuse{bbl@xeclasse@bbl@tempb @#2}%
5249 \bbl@ifunset{bbl@xeclasse@bbl@tempb @#2}{#2}} %
5250 = \expandafter{%
5251 \csname bbl@ic@bbl@kv@label @#2\expandafter\endcsname
5252 \csname\zap@space bbl@xeinter@bbl@kv@label
5253 @#3@#4@#2 \@empty\endcsname}}}}
5254 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5255 \bbl@ifunset{bbl@ic@#1@language}%
5256 {\bbl@error{unknown-interchar}{#1}}}%
5257 {\bbl@csarg\let{ic@#1@language}\@firstofone}}
5258 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5259 \bbl@ifunset{bbl@ic@#1@language}%
5260 {\bbl@error{unknown-interchar-b}{#1}}}%
5261 {\bbl@csarg\let{ic@#1@language}\@gobble}}
5262 </xetex>

```

10.3. Layout

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

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

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

```

5263 < *xetex | texxet >
5264 \providecommand\bbl@provide@intraspace{}
5265 \bbl@trace{Redefinitions for bidi layout}

    Finish here if there in no layout.

5266 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5267 \ifx\bbl@opt@layout\@undefined\else % Plain
5268 \IfBabelLayout{nopars}
5269 {}
5270 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5271 \fi
5272 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5273 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5274 \ifnum\bbl@bidimode>\z@
5275 \IfBabelLayout{pars}
5276 {\def\hangfrom#1{%
5277 \setbox\@tempboxa\hbox{#1}}%
5278 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5279 \noindent\box\@tempboxa}
5280 \def\raggedright{%
5281 \let\@centercr
5282 \bbl@startskip\z@skip
5283 \@rightskip\@flushglue
5284 \bbl@endskip\@rightskip
5285 \parindent\z@
5286 \parfillskip\bbl@startskip}
5287 \def\raggedleft{%
5288 \let\@centercr
5289 \bbl@startskip\@flushglue
5290 \bbl@endskip\z@skip

```

```

5291 \parindent\z@
5292 \parfillskip\bbl@endskip}}
5293 {}
5294 \fi
5295 \IfBabelLayout{lists}
5296 {\bbl@sreplace\list
5297 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5298 \def\bbl@listleftmargin{%
5299 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5300 \ifcase\bbl@engine
5301 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5302 \def\p@enumiii{\p@enumii}\theenumii{}\%
5303 \fi
5304 \bbl@sreplace\@verbatim
5305 {\leftskip\@totalleftmargin}%
5306 {\bbl@startskip\textwidth
5307 \advance\bbl@startskip-\linewidth}%
5308 \bbl@sreplace\@verbatim
5309 {\rightskip\z@skip}%
5310 {\bbl@endskip\z@skip}}%
5311 {}
5312 \IfBabelLayout{contents}
5313 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5314 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5315 {}
5316 \IfBabelLayout{columns}
5317 {\bbl@sreplace\@outputdblcol{\hbxt@\textwidth}{\bbl@outputbox}%
5318 \def\bbl@outputbox#1{%
5319 \hbxt@\textwidth{%
5320 \hskip\columnwidth
5321 \hfil
5322 {\normalcolor\vrule \@width\columnseprule}%
5323 \hfil
5324 \hbxt@\columnwidth{\box\@leftcolumn \hss}%
5325 \hskip-\textwidth
5326 \hbxt@\columnwidth{\box\@outputbox \hss}%
5327 \hskip\columnsep
5328 \hskip\columnwidth}}}%
5329 {}

```

Implicitly reverses sectioning labels in `bidibasic`, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```

5330 \IfBabelLayout{counters*}%
5331 {\bbl@add\bbl@opt@layout{.counters.}%
5332 \AddToHook{shipout/before}{%
5333 \let\bbl@tempa\babelsublr
5334 \let\babelsublr\@firstofone
5335 \let\bbl@save@thepage\thepage
5336 \protected@edef\thepage{\thepage}%
5337 \let\babelsublr\bbl@tempa}%
5338 \AddToHook{shipout/after}{%
5339 \let\thepage\bbl@save@thepage}}{}
5340 \IfBabelLayout{counters}%
5341 {\let\bbl@latinarabic=\@arabic
5342 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5343 \let\bbl@asciroman=\@roman
5344 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5345 \let\bbl@asciiRoman=\@Roman
5346 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5347 \fi % end if layout
5348 /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).

```
5349 < *texxet>
5350 \def\bbl@provide@extra#1{%
5351   % == auto-select encoding ==
5352   \ifx\bbl@encoding@select@off\@empty\else
5353     \bbl@ifunset\bbl@encoding@#1{%
5354       {\def\elt##1{,##1,}%
5355        \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5356        \count@z@
5357        \bbl@foreach\bbl@tempe{%
5358          \def\bbl@tempd{##1}% Save last declared
5359          \advance\count@\@ne}%
5360        \ifnum\count@>\@ne % (1)
5361          \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5362          \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5363          \bbl@replace\bbl@tempa{ }{,}%
5364          \global\bbl@csarg\let{encoding@#1}\@empty
5365          \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5366          \ifin\@else % if main encoding included in ini, do nothing
5367            \let\bbl@tempb\relax
5368            \bbl@foreach\bbl@tempa{%
5369              \ifx\bbl@tempb\relax
5370                \bbl@xin@{,##1,}{,\bbl@tempe,}%
5371                \ifin\def\bbl@tempb{##1}\fi
5372              \fi}%
5373            \ifx\bbl@tempb\relax\else
5374              \bbl@exp{%
5375                \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5376                \gdef\<bbl@encoding@#1>{%
5377                  \\babel@save\\f@encoding
5378                  \\bbl@add\\originalTeX{\\selectfont}%
5379                  \\fontencoding{\bbl@tempb}%
5380                  \\selectfont}}%
5381              \fi
5382            \fi
5383          \fi}%
5384        }%
5385      \fi}
5386 < /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).

```

5387 (*luatex)
5388 \directlua{ Babel = Babel or {} } % DL2
5389 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5390 \bbl@trace{Read language.dat}
5391 \ifx\bbl@readstream\undefined
5392 \csname newread\endcsname\bbl@readstream
5393 \fi
5394 \begingroup
5395 \toks@{}
5396 \count@\z@ % 0=start, 1=0th, 2=normal
5397 \def\bbl@process@line#1#2 #3 #4 {%
5398 \ifx=#1%
5399 \bbl@process@synonym{#2}%
5400 \else
5401 \bbl@process@language{#1#2}{#3}{#4}%
5402 \fi
5403 \ignorespaces}
5404 \def\bbl@manylang{%
5405 \ifnum\bbl@last>\@ne
5406 \bbl@info{Non-standard hyphenation setup}%
5407 \fi
5408 \let\bbl@manylang\relax}
5409 \def\bbl@process@language#1#2#3{%
5410 \ifcase\count@
5411 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5412 \or
5413 \count@\tw@
5414 \fi
5415 \ifnum\count@=\tw@
5416 \expandafter\addlanguage\csname l@#1\endcsname
5417 \language\allocationnumber
5418 \chardef\bbl@last\allocationnumber
5419 \bbl@manylang
5420 \let\bbl@elt\relax
5421 \xdef\bbl@languages{%
5422 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5423 \fi
5424 \the\toks@
5425 \toks@{}}
5426 \def\bbl@process@synonym@aux#1#2{%
5427 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5428 \let\bbl@elt\relax
5429 \xdef\bbl@languages{%
5430 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5431 \def\bbl@process@synonym#1{%
5432 \ifcase\count@
5433 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5434 \or

```

```

5435 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5436 \else
5437 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5438 \fi}
5439 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5440 \chardef\l@english\z@
5441 \chardef\l@USenglish\z@
5442 \chardef\bbl@last\z@
5443 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}{}
5444 \gdef\bbl@languages{%
5445 \bbl@elt{english}{0}{hyphen.tex}}{}%
5446 \bbl@elt{USenglish}{0}{}{}
5447 \else
5448 \global\let\bbl@languages@format\bbl@languages
5449 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5450 \ifnum#2>\z@\else
5451 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5452 \fi}%
5453 \xdef\bbl@languages{\bbl@languages}%
5454 \fi
5455 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}}{} % Define flags
5456 \bbl@languages
5457 \openin\bbl@readstream=language.dat
5458 \ifeof\bbl@readstream
5459 \bbl@warning{I couldn't find language.dat. No additional\\%
5460 patterns loaded. Reported}%
5461 \else
5462 \loop
5463 \endlinechar\m@ne
5464 \read\bbl@readstream to \bbl@line
5465 \endlinechar\^^M
5466 \if T\ifeof\bbl@readstream F\fi T\relax
5467 \ifx\bbl@line\@empty\else
5468 \edef\bbl@line{\bbl@line\space\space\space}%
5469 \expandafter\bbl@process@line\bbl@line\relax
5470 \fi
5471 \repeat
5472 \fi
5473 \closein\bbl@readstream
5474 \endgroup
5475 \bbl@trace{Macros for reading patterns files}
5476 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5477 \ifx\babelcatcodetablenum\undefined
5478 \ifx\newcatcodetable\undefined
5479 \def\babelcatcodetablenum{5211}
5480 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5481 \else
5482 \newcatcodetable\babelcatcodetablenum
5483 \newcatcodetable\bbl@pattcodes
5484 \fi
5485 \else
5486 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5487 \fi
5488 \def\bbl@luapatterns#1#2{%
5489 \bbl@get@enc#1::\@@@
5490 \setbox\z@\hbox\bgroup
5491 \begingroup
5492 \savecatcodetable\babelcatcodetablenum\relax
5493 \initcatcodetable\bbl@pattcodes\relax
5494 \catcodetable\bbl@pattcodes\relax
5495 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5496 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5497 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12

```

```

5498      \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5499      \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5500      \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
5501      \input #1\relax
5502      \catcodetable\babelcatcodetablenum\relax
5503      \endgroup
5504      \def\bbl@tempa{#2}%
5505      \ifx\bbl@tempa\@empty\else
5506          \input #2\relax
5507      \fi
5508      \egroup}%
5509 \def\bbl@patterns@lua#1{%
5510     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5511         \csname l@#1\endcsname
5512         \edef\bbl@tempa{#1}%
5513     \else
5514         \csname l@#1:\f@encoding\endcsname
5515         \edef\bbl@tempa{#1:\f@encoding}%
5516     \fi\relax
5517     \namedef{lu@texhyphen@loaded@the\language}{}% Temp
5518     \ifundefined{bbl@hyphendata@the\language}%
5519         {\def\bbl@elt##1##2###4{%
5520             \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5521             \def\bbl@tempb{##3}%
5522             \ifx\bbl@tempb\@empty\else % if not a synonymous
5523                 \def\bbl@tempc{{##3}{##4}}%
5524             \fi
5525             \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5526             \fi}%
5527     \bbl@languages
5528     \ifundefined{bbl@hyphendata@the\language}%
5529         {\bbl@info{No hyphenation patterns were set for\%
5530             language '\bbl@tempa'. Reported}}%
5531         {\expandafter\expandafter\expandafter\bbl@luapatterns
5532             \csname bbl@hyphendata@the\language\endcsname}}}%
5533 \endinput\fi

```

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

```

5534 \ifx\DisableBabelHook\@undefined
5535     \AddBabelHook{luatex}{everylanguage}{%
5536         \def\process@language##1##2##3{%
5537             \def\process@line####1####2 ####3 ####4 {}}%
5538     \AddBabelHook{luatex}{loadpatterns}{%
5539         \input #1\relax
5540         \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5541             {{#1}}}%
5542     \AddBabelHook{luatex}{loadexceptions}{%
5543         \input #1\relax
5544         \def\bbl@tempb##1##2{{##1}{##2}}%
5545         \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5546             {\expandafter\expandafter\expandafter\bbl@tempb
5547                 \csname bbl@hyphendata@the\language\endcsname}}%
5548     \endinput\fi

```

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

```

5549 \begin{group}
5550 \catcode`\%=12
5551 \catcode`\'=12
5552 \catcode`\\"=12
5553 \catcode`\:=12
5554 \directlua{
5555     Babel.locale_props = Babel.locale_props or {}
5556     function Babel.lua_error(e, a)

```

```

5557     tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5558         e .. '}' .. (a or '') .. '}'})
5559 end
5560
5561 function Babel.bytes(line)
5562     return line:gsub("(.)",
5563         function (chr) return unicode.utf8.char(string.byte(chr)) end)
5564 end
5565
5566 function Babel.priority_in_callback(name,description)
5567     for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5568         if v == description then return i end
5569     end
5570     return false
5571 end
5572
5573 function Babel.begin_process_input()
5574     if luatexbase and luatexbase.add_to_callback then
5575         luatexbase.add_to_callback('process_input_buffer',
5576             Babel.bytes, 'Babel.bytes')
5577     else
5578         Babel.callback = callback.find('process_input_buffer')
5579         callback.register('process_input_buffer', Babel.bytes)
5580     end
5581 end
5582 function Babel.end_process_input ()
5583     if luatexbase and luatexbase.remove_from_callback then
5584         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5585     else
5586         callback.register('process_input_buffer', Babel.callback)
5587     end
5588 end
5589
5590 function Babel.str_to_nodes(fn, matches, base)
5591     local n, head, last
5592     if fn == nil then return nil end
5593     for s in string.utfvalues(fn(matches)) do
5594         if base.id == 7 then
5595             base = base.replace
5596         end
5597         n = node.copy(base)
5598         n.char = s
5599         if not head then
5600             head = n
5601         else
5602             last.next = n
5603         end
5604         last = n
5605     end
5606     return head
5607 end
5608
5609 Babel.linebreaking = Babel.linebreaking or {}
5610 Babel.linebreaking.before = {}
5611 Babel.linebreaking.after = {}
5612 Babel.locale = {}
5613 function Babel.linebreaking.add_before(func, pos)
5614     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5615     if pos == nil then
5616         table.insert(Babel.linebreaking.before, func)
5617     else
5618         table.insert(Babel.linebreaking.before, pos, func)
5619     end

```

```

5620 end
5621 function Babel.linebreaking.add_after(func)
5622   tex.print([[\\noexpand\\csname bbl@lua hyphenate\\endcsname]])
5623   table.insert(Babel.linebreaking.after, func)
5624 end
5625
5626 function Babel.addpatterns(pp, lg)
5627   local lg = lang.new(lg)
5628   local pats = lang.patterns(lg) or ''
5629   lang.clear_patterns(lg)
5630   for p in pp:gmatch('[^%s]+') do
5631     ss = ''
5632     for i in string.utfcharacters(p:gsub('%d', '')) do
5633       ss = ss .. '%d?' .. i
5634     end
5635     ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5636     ss = ss:gsub('%.%d%?$', '%%.')
5637     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5638     if n == 0 then
5639       tex.sprint(
5640         [[\\string\\csname\\space bbl@info\\endcsname{New pattern: }
5641         .. p .. [{}]]])
5642       pats = pats .. ' ' .. p
5643     else
5644       tex.sprint(
5645         [[\\string\\csname\\space bbl@info\\endcsname{Renew pattern: }
5646         .. p .. [{}]]])
5647     end
5648   end
5649   lang.patterns(lg, pats)
5650 end
5651
5652 Babel.characters = Babel.characters or {}
5653 Babel.ranges = Babel.ranges or {}
5654 function Babel.hlist_has_bidi(head)
5655   local has_bidi = false
5656   local ranges = Babel.ranges
5657   for item in node.traverse(head) do
5658     if item.id == node.id'glyph' then
5659       local itemchar = item.char
5660       local chardata = Babel.characters[itemchar]
5661       local dir = chardata and chardata.d or nil
5662       if not dir then
5663         for nn, et in ipairs(ranges) do
5664           if itemchar < et[1] then
5665             break
5666           elseif itemchar <= et[2] then
5667             dir = et[3]
5668             break
5669           end
5670         end
5671       end
5672       if dir and (dir == 'al' or dir == 'r') then
5673         has_bidi = true
5674       end
5675     end
5676   end
5677   return has_bidi
5678 end
5679 function Babel.set_chranges_b (script, chrng)
5680   if chrng == '' then return end
5681   texio.write('Replacing ' .. script .. ' script ranges')
5682   Babel.script_blocks[script] = {}

```

```

5683     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5684         table.insert(
5685             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5686     end
5687 end
5688
5689 function Babel.discard_sublr(str)
5690     if str:find( [[\string\indexentry]] ) and
5691         str:find( [[\string\babelsublr]] ) then
5692         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5693             function(m) return m:sub(2,-2) end )
5694     end
5695     return str
5696 end
5697 }
5698 \endgroup
5699 \ifx\newattribute\undefined\else % Test for plain
5700     \newattribute\bbl@attr@locale % DL4
5701     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5702     \AddBabelHook{luatex}{beforeextras}{%
5703         \setattribute\bbl@attr@locale\localeid}
5704 \fi
5705 %
5706 \def\BabelStringsDefault{unicode}
5707 \let\luabbl@stop\relax
5708 \AddBabelHook{luatex}{encodedcommands}{%
5709     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5710     \ifx\bbl@tempa\bbl@tempb\else
5711         \directlua{Babel.begin_process_input()}%
5712         \def\luabbl@stop{%
5713             \directlua{Babel.end_process_input()}}%
5714     \fi}%
5715 \AddBabelHook{luatex}{stopcommands}{%
5716     \luabbl@stop
5717     \let\luabbl@stop\relax}
5718 %
5719 \AddBabelHook{luatex}{patterns}{%
5720     \ifundefined{bbl@hyphendata@the\language}%
5721     {\def\bbl@elt##1##2##3##4{%
5722         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5723         \def\bbl@tempb{##3}%
5724         \ifx\bbl@tempb\@empty\else % if not a synonymous
5725             \def\bbl@tempc{##3}{##4}%
5726         \fi
5727         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5728         \fi}%
5729     \bbl@languages
5730     \ifundefined{bbl@hyphendata@the\language}%
5731     {\bbl@info{No hyphenation patterns were set for\%
5732         language '#2'. Reported}}%
5733     {\expandafter\expandafter\expandafter\bbl@luapatterns
5734         \csname bbl@hyphendata@the\language\endcsname}}}%
5735 \ifundefined{bbl@patterns@}{}%
5736     \begingroup
5737         \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5738         \ifin@else
5739             \ifx\bbl@patterns@\@empty\else
5740                 \directlua{ Babel.addpatterns(
5741                     [[\bbl@patterns@]], \number\language) }%
5742             \fi
5743             \ifundefined{bbl@patterns@#1}%
5744                 \@empty
5745                 {\directlua{ Babel.addpatterns(

```

```

5746          [[\space\csname bbl@patterns@#1\endcsname]],
5747          \number\language) } }%
5748      \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5749      \fi
5750  \endgroup}%
5751  \bbl@exp{%
5752      \bbl@ifunset{\bbl@prehc@language}{%
5753          {\bbl@ifblank{\bbl@cs{\bbl@prehc@language}}{}}%
5754          {\prehyphenchar=\bbl@cl{\bbl@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.

```

5755 \@onlypreamble\babelpatterns
5756 \AtEndOfPackage{%
5757   \newcommand\babelpatterns[2][\@empty]{%
5758     \ifx\bbl@patterns@relax
5759       \let\bbl@patterns@empty
5760     \fi
5761     \ifx\bbl@pttnlist@empty\else
5762       \bbl@warning{%
5763         You must not intermingle \string\selectlanguage\space and\%
5764         \string\babelpatterns\space or some patterns will not\%
5765         be taken into account. Reported}%
5766       \fi
5767       \ifx@empty#1%
5768         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5769       \else
5770         \edef\bbl@tempb{\zap@space#1 \@empty}%
5771         \bbl@for\bbl@tempa\bbl@tempb{%
5772           \bbl@fixname\bbl@tempa
5773           \bbl@iflanguage\bbl@tempa{%
5774             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5775               \@ifundefined{\bbl@patterns@\bbl@tempa}%
5776               \@empty
5777               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5778             #2}}}%
5779         \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.

```

5780 \def\bbl@intraspace#1 #2 #3\@{%
5781   \directlua{
5782     Babel.intraspaces = Babel.intraspaces or {}
5783     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5784       {b = #1, p = #2, m = #3}
5785     Babel.locale_props[\the\localeid].intraspace = %
5786       {b = #1, p = #2, m = #3}
5787   }}
5788 \def\bbl@intrapenalty#1\@{%
5789   \directlua{
5790     Babel.intrapenalties = Babel.intrapenalties or {}
5791     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5792     Babel.locale_props[\the\localeid].intrapenalty = #1
5793   }}
5794 \begingroup
5795 \catcode`\%=12
5796 \catcode`\&=14

```

```

5797 \catcode\'=12
5798 \catcode\-=12
5799 \gdef\bbl@seaintraspace{&
5800 \let\bbl@seaintraspace\relax
5801 \directlua{
5802   Babel.sea_enabled = true
5803   Babel.sea_ranges = Babel.sea_ranges or {}
5804   function Babel.set_chranges (script, chrng)
5805     local c = 0
5806     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5807       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5808       c = c + 1
5809     end
5810   end
5811   function Babel.sea_disc_to_space (head)
5812     local sea_ranges = Babel.sea_ranges
5813     local last_char = nil
5814     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5815     for item in node.traverse(head) do
5816       local i = item.id
5817       if i == node.id'glyph' then
5818         last_char = item
5819       elseif i == 7 and item.subtype == 3 and last_char
5820         and last_char.char > 0x0C99 then
5821         quad = font.getfont(last_char.font).size
5822         for lg, rg in pairs(sea_ranges) do
5823           if last_char.char > rg[1] and last_char.char < rg[2] then
5824             lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5825             local intraspace = Babel.intraspaces[lg]
5826             local intrapenalty = Babel.intrapenalties[lg]
5827             local n
5828             if intrapenalty ~= 0 then
5829               n = node.new(14, 0)      &% penalty
5830               n.penalty = intrapenalty
5831               node.insert_before(head, item, n)
5832             end
5833             n = node.new(12, 13)      &% (glue, spaceskip)
5834             node.setglue(n, intraspace.b * quad,
5835               intraspace.p * quad,
5836               intraspace.m * quad)
5837             node.insert_before(head, item, n)
5838             node.remove(head, item)
5839           end
5840         end
5841       end
5842     end
5843   end
5844 }&
5845 \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.

```

5846 \catcode\%=14
5847 \gdef\bbl@cjkintraspacespace{%
5848 \let\bbl@cjkintraspacespace\relax
5849 \directlua{
5850   require('babel-data-cjk.lua')

```

```

5851 Babel.cjk_enabled = true
5852 function Babel.cjk_linebreak(head)
5853     local GLYPH = node.id'glyph'
5854     local last_char = nil
5855     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5856     local last_class = nil
5857     local last_lang = nil
5858     for item in node.traverse(head) do
5859         if item.id == GLYPH then
5860             local lang = item.lang
5861             local LOCALE = node.get_attribute(item,
5862                 Babel.attr_locale)
5863             local props = Babel.locale_props[LOCALE] or {}
5864             local class = Babel.cjk_class[item.char].c
5865             if props.cjk_quotes and props.cjk_quotes[item.char] then
5866                 class = props.cjk_quotes[item.char]
5867             end
5868             if class == 'cp' then class = 'cl' % ) as CL
5869             elseif class == 'id' then class = 'I'
5870             elseif class == 'cj' then class = 'I' % loose
5871             end
5872             local br = 0
5873             if class and last_class and Babel.cjk_breaks[last_class][class] then
5874                 br = Babel.cjk_breaks[last_class][class]
5875             end
5876             if br == 1 and props.linebreak == 'c' and
5877                 lang ~= \the\l@nohyphenation\space and
5878                 last_lang ~= \the\l@nohyphenation then
5879                 local intrapenalty = props.intrapenalty
5880                 if intrapenalty ~= 0 then
5881                     local n = node.new(14, 0)      % penalty
5882                     n.penalty = intrapenalty
5883                     node.insert_before(head, item, n)
5884                 end
5885                 local intraspace = props.intraspace
5886                 local n = node.new(12, 13)      % (glue, spaceskip)
5887                 node.setglue(n, intraspace.b * quad,
5888                     intraspace.p * quad,
5889                     intraspace.m * quad)
5890                 node.insert_before(head, item, n)
5891             end
5892             if font.getfont(item.font) then
5893                 quad = font.getfont(item.font).size
5894             end
5895             last_class = class
5896             last_lang = lang
5897             else % if penalty, glue or anything else
5898                 last_class = nil
5899             end
5900         end
5901         lang.hyphenate(head)
5902     end
5903 }%
5904 \bbl@luahyphenate}
5905 \gdef\bbl@luahyphenate{%
5906 \let\bbl@luahyphenate\relax
5907 \directlua{
5908     luatexbase.add_to_callback('hyphenate',
5909     function (head, tail)
5910         if Babel.linebreaking.before then
5911             for k, func in ipairs(Babel.linebreaking.before) do
5912                 func(head)
5913             end

```

```

5914     end
5915     lang.hyphenate(head)
5916     if Babel.cjk_enabled then
5917         Babel.cjk_linebreak(head)
5918     end
5919     if Babel.linebreaking.after then
5920         for k, func in ipairs(Babel.linebreaking.after) do
5921             func(head)
5922         end
5923     end
5924     if Babel.set_hboxed then
5925         Babel.set_hboxed(head)
5926     end
5927     if Babel.sea_enabled then
5928         Babel.sea_disc_to_space(head)
5929     end
5930 end,
5931 'Babel.hyphenate')
5932 }}
5933 \endgroup
5934 %
5935 \def\bbl@provide@intraspace{%
5936   \bbl@ifunset\bbl@intsp@\languagename{}\}%
5937   {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5938     \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}}%
5939   \ifin@           % cjk
5940     \bbl@cjkintraspace
5941     \directlua{
5942       Babel.locale_props = Babel.locale_props or {}
5943       Babel.locale_props[\the\localeid].linebreak = 'c'
5944     }%
5945     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5946     \ifx\bbl@KVP@intrapenalty\@nnil
5947       \bbl@intrapenalty0\@
5948     \fi
5949   \else           % sea
5950     \bbl@seaintraspace
5951     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5952     \directlua{
5953       Babel.sea_ranges = Babel.sea_ranges or {}
5954       Babel.set_chranges('\bbl@cl{sbcpr}',
5955         '\bbl@cl{chrng}')
5956     }%
5957     \ifx\bbl@KVP@intrapenalty\@nnil
5958       \bbl@intrapenalty0\@
5959     \fi
5960   \fi
5961 \fi
5962 \ifx\bbl@KVP@intrapenalty\@nnil\else
5963   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5964 \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`.

```

5965 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5966 \def\bblar@chars{%
5967   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5968   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5969   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5970 \def\bblar@elongated{%
5971   0626,0628,062A,062B,0633,0634,0635,0636,063B,%

```

```

5972 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5973 0649,064A}
5974 \begingroup
5975 \catcode`_ =11 \catcode`:=11
5976 \gdef\bblar@nofswarn{\gdef/msg_warning:nx##1##2##3{}}
5977 \endgroup
5978 \gdef\bbl@arabicjust{%
5979 \let\bbl@arabicjust\relax
5980 \newattribute\bblar@kashida
5981 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5982 \bblar@kashida=\z@
5983 \bbl@patchfont{\bbl@parsejalt}}%
5984 \directlua{
5985   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5986   Babel.arabic.elong_map[\the\localeid] = {}
5987   luatexbase.add_to_callback('post_linebreak_filter',
5988     Babel.arabic.justify, 'Babel.arabic.justify')
5989   luatexbase.add_to_callback('hpack_filter',
5990     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5991 }%

```

Save both node lists to make replacement.

```

5992 \def\bblar@fetchjalt#1#2#3#4{%
5993 \bbl@exp{\bbl@foreach{#1}}{%
5994 \bbl@ifunset\bblar@JE@##1}%
5995   {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5996   {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5997 \directlua{%
5998   local last = nil
5999   for item in node.traverse(tex.box[0].head) do
6000     if item.id == node.id'glyph' and item.char > 0x600 and
6001       not (item.char == 0x200D) then
6002       last = item
6003     end
6004   end
6005   Babel.arabic.#3['##1#4'] = last.char
6006 }}}

```

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

```

6007 \gdef\bbl@parsejalt{%
6008 \ifx\addfontfeature\undefined\else
6009 \bbl@xin@{/e}{\bbl@cl{\lnbrk}}%
6010 \ifin@
6011 \directlua{%
6012   if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
6013     Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
6014     tex.print([[string\csname\space bbl@parsejalti\endcsname]])
6015   end
6016 }%
6017 \fi
6018 \fi}
6019 \gdef\bbl@parsejalti{%
6020 \begingroup
6021 \let\bbl@parsejalt\relax % To avoid infinite loop
6022 \edef\bbl@tempb{\fontid\font}%
6023 \bblar@nofswarn
6024 \@nameuse{tracinglostchars}\z@
6025 \bblar@fetchjalt\bblar@elongated{}{from}{}%
6026 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
6027 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
6028 \addfontfeature{RawFeature+=jalt}%
6029 % \namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
6030 \bblar@fetchjalt\bblar@elongated{}{dest}{}%

```

```

6031 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
6032 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
6033 \directlua{%
6034     for k, v in pairs(Babel.arabic.from) do
6035         if Babel.arabic.dest[k] and
6036             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
6037             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
6038                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
6039         end
6040     end
6041 }%
6042 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

6043 \beginngroup
6044 \catcode`#=11
6045 \catcode`~=11
6046 \directlua{
6047
6048 Babel.arabic = Babel.arabic or {}
6049 Babel.arabic.from = {}
6050 Babel.arabic.dest = {}
6051 Babel.arabic.justify_factor = 0.95
6052 Babel.arabic.justify_enabled = true
6053 Babel.arabic.kashida_limit = -1
6054
6055 function Babel.arabic.justify(head)
6056     if not Babel.arabic.justify_enabled then return head end
6057     for line in node.traverse_id(node.id'hlist', head) do
6058         Babel.arabic.justify_hlist(head, line)
6059     end
6060     % In case the very first item is a line (eg, in \vbox):
6061     while head.prev do head = head.prev end
6062     return head
6063 end
6064
6065 function Babel.arabic.justify_hbox(head, gc, size, pack)
6066     local has_inf = false
6067     if Babel.arabic.justify_enabled and pack == 'exactly' then
6068         for n in node.traverse_id(12, head) do
6069             if n.stretch_order > 0 then has_inf = true end
6070         end
6071         if not has_inf then
6072             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
6073         end
6074     end
6075     return head
6076 end
6077
6078 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
6079     local d, new
6080     local k_list, k_item, pos_inline
6081     local width, width_new, full, k_curr, wt_pos, goal, shift
6082     local subst_done = false
6083     local elong_map = Babel.arabic.elong_map
6084     local cnt
6085     local last_line
6086     local GLYPH = node.id'glyph'
6087     local KASHIDA = Babel.attr_kashida
6088     local LOCALE = Babel.attr_locale
6089
6090     if line == nil then
6091         line = {}

```

```

6092     line.glue_sign = 1
6093     line.glue_order = 0
6094     line.head = head
6095     line.shift = 0
6096     line.width = size
6097 end
6098
6099 % Exclude last line.
6100 if ((line.next ~= nil or line.glue_sign == 1) and line.glue_order == 0) then
6101     elongs = {}      % Stores elongated candidates of each line
6102     k_list = {}      % And all letters with kashida
6103     pos_inline = 0   % Not yet used
6104
6105     for n in node.traverse_id(GLYPH, line.head) do
6106         pos_inline = pos_inline + 1 % To find where it is. Not used.
6107
6108         % Elongated glyphs
6109         if elong_map then
6110             local locale = node.get_attribute(n, LOCALE)
6111             if elong_map[locale] and elong_map[locale][n.font] and
6112                 elong_map[locale][n.font][n.char] then
6113                 table.insert(elongs, {node = n, locale = locale} )
6114                 node.set_attribute(n.prev, KASHIDA, 0)
6115             end
6116         end
6117
6118         % Tatwil. First create a list of nodes marked with kashida. The
6119         % rest of nodes can be ignored. The list of used weights is build
6120         % when transforms with the key kashida= are declared.
6121         if Babel.kashida_wts then
6122             local k_wt = node.get_attribute(n, KASHIDA)
6123             if k_wt > 0 then % todo. parameter for multi inserts
6124                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
6125             end
6126         end
6127
6128     end % of node.traverse_id
6129
6130     if #elongs == 0 and #k_list == 0 then goto next_line end
6131     full = line.width
6132     shift = line.shift
6133     goal = full * Babel.arabic.justify_factor % A bit crude
6134     width = node.dimensions(line.head) % The 'natural' width
6135
6136     % == Elongated ==
6137     % Original idea taken from 'chickenize'
6138     while (#elongs > 0 and width < goal) do
6139         subst_done = true
6140         local x = #elongs
6141         local curr = elongs[x].node
6142         local oldchar = curr.char
6143         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6144         width = node.dimensions(line.head) % Check if the line is too wide
6145         % Substitute back if the line would be too wide and break:
6146         if width > goal then
6147             curr.char = oldchar
6148             break
6149         end
6150         % If continue, pop the just substituted node from the list:
6151         table.remove(elongs, x)
6152     end
6153
6154     % == Tatwil ==

```

```

6155 % Traverse the kashida node list so many times as required, until
6156 % the line is filled. The first pass adds a tatweel after each
6157 % node with kashida in the line, the second pass adds another one,
6158 % and so on. In each pass, add first the kashida with the highest
6159 % weight, then with lower weight and so on.
6160 if #k_list == 0 then goto next_line end
6161
6162 width = node.dimensions(line.head) % The 'natural' width
6163 k_curr = #k_list % Traverse backwards, from the end
6164 wt_pos = 1
6165
6166 while width < goal do
6167     subst_done = true
6168     k_item = k_list[k_curr].node
6169     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6170         d = node.copy(k_item)
6171         d.char = 0x0640
6172         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6173         d.xoffset = 0
6174         line.head, new = node.insert_after(line.head, k_item, d)
6175         width_new = node.dimensions(line.head)
6176         if width > goal or width == width_new then
6177             node.remove(line.head, new) % Better compute before
6178             break
6179         end
6180         if Babel.fix_diacr then
6181             Babel.fix_diacr(k_item.next)
6182         end
6183         width = width_new
6184     end
6185     if k_curr == 1 then
6186         k_curr = #k_list
6187         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6188     else
6189         k_curr = k_curr - 1
6190     end
6191 end
6192
6193 % Limit the number of tatweel by removing them. Not very efficient,
6194 % but it does the job in a quite predictable way.
6195 if Babel.arabic.kashida_limit > -1 then
6196     cnt = 0
6197     for n in node.traverse_id(GLYPH, line.head) do
6198         if n.char == 0x0640 then
6199             cnt = cnt + 1
6200             if cnt > Babel.arabic.kashida_limit then
6201                 node.remove(line.head, n)
6202             end
6203         else
6204             cnt = 0
6205         end
6206     end
6207 end
6208
6209 ::next_line::
6210
6211 % Must take into account marks and ins, see luatex manual.
6212 % Have to be executed only if there are changes. Investigate
6213 % what's going on exactly.
6214 if subst_done and not gc then
6215     d = node.hpack(line.head, full, 'exactly')
6216     d.shift = shift
6217     node.insert_before(head, line, d)

```

```

6218      node.remove(head, line)
6219    end
6220  end % if process line
6221 end
6222 }
6223 \endgroup
6224 \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`.

```

6225 \def\bbl@scr@node@list{%
6226   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6227   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6228 \ifnum\bbl@bidimode=102 % bidi-r
6229   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6230 \fi
6231 \def\bbl@set@renderer{%
6232   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6233   \ifin@
6234     \let\bbl@unset@renderer\relax
6235   \else
6236     \bbl@exp{%
6237       \def\\bbl@unset@renderer{%
6238         \def<g__fontspec_default_fontopts_clist>{%
6239           \[g__fontspec_default_fontopts_clist]}}%
6240       \def<g__fontspec_default_fontopts_clist>{%
6241         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
6242     \fi}
6243 <@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.

```

6244 \directlua{% DL6
6245   Babel.script_blocks = {
6246     ['dflt'] = {},
6247     ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6248               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFD}, {0x1EE00, 0x1EEFF}},
6249     ['Armn'] = {{0x0530, 0x058F}},
6250     ['Beng'] = {{0x0980, 0x09FF}},
6251     ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6252     ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6253     ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6254               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6255     ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6256     ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6257               {0xAB00, 0xAB2F}},

```

```

6258 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6259 % Don't follow strictly Unicode, which places some Coptic letters in
6260 % the 'Greek and Coptic' block
6261 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6262 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6263             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6264             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6265             {0x2000, 0x2A6DF}, {0x2A700, 0x2B73F},
6266             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6267             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6268 ['Hebr'] = {{0x0590, 0x05FF},
6269             {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6270 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6271             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6272 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6273 ['Knda'] = {{0x0C80, 0x0CFF}},
6274 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6275             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6276             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6277 ['Lao'] = {{0x0E80, 0x0EFF}},
6278 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6279             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6280             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6281 ['Mahj'] = {{0x11150, 0x1117F}},
6282 ['Mlym'] = {{0x0D00, 0x0D7F}},
6283 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6284 ['Orya'] = {{0x0B00, 0x0B7F}},
6285 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6286 ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6287 ['Taml'] = {{0x0B80, 0x0BFF}},
6288 ['Telu'] = {{0x0C00, 0x0C7F}},
6289 ['Tfng'] = {{0x2D30, 0x2D7F}},
6290 ['Thai'] = {{0x0E00, 0x0E7F}},
6291 ['Tibt'] = {{0x0F00, 0x0FFF}},
6292 ['Vaii'] = {{0xA500, 0xA63F}},
6293 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6294 }
6295
6296 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6297 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6298 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6299
6300 function Babel.locale_map(head)
6301   if not Babel.locale_mapped then return head end
6302
6303   local LOCALE = Babel.attr_locale
6304   local GLYPH = node.id('glyph')
6305   local inmath = false
6306   local toloc_save
6307   for item in node.traverse(head) do
6308     local toloc
6309     if not inmath and item.id == GLYPH then
6310       % Optimization: build a table with the chars found
6311       if Babel.chr_to_loc[item.char] then
6312         toloc = Babel.chr_to_loc[item.char]
6313       else
6314         for lc, maps in pairs(Babel.loc_to_scr) do
6315           for _, rg in pairs(maps) do
6316             if item.char >= rg[1] and item.char <= rg[2] then
6317               Babel.chr_to_loc[item.char] = lc
6318               toloc = lc
6319               break
6320             end

```

```

6321         end
6322     end
6323     % Treat composite chars in a different fashion, because they
6324     % 'inherit' the previous locale.
6325     if (item.char >= 0x0300 and item.char <= 0x036F) or
6326        (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6327        (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6328         Babel.chr_to_loc[item.char] = -2000
6329         toloc = -2000
6330     end
6331     if not toloc then
6332         Babel.chr_to_loc[item.char] = -1000
6333     end
6334     end
6335     if toloc == -2000 then
6336         toloc = toloc_save
6337     elseif toloc == -1000 then
6338         toloc = nil
6339     end
6340     if toloc and Babel.locale_props[toloc] and
6341        Babel.locale_props[toloc].letters and
6342        tex.getcatcode(item.char) \string~= 11 then
6343         toloc = nil
6344     end
6345     if toloc and Babel.locale_props[toloc].script
6346        and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6347        and Babel.locale_props[toloc].script ==
6348        Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6349         toloc = nil
6350     end
6351     if toloc then
6352         if Babel.locale_props[toloc].lg then
6353             item.lang = Babel.locale_props[toloc].lg
6354             node.set_attribute(item, LOCALE, toloc)
6355         end
6356         if Babel.locale_props[toloc]['/'..item.font] then
6357             item.font = Babel.locale_props[toloc]['/'..item.font]
6358         end
6359     end
6360     toloc_save = toloc
6361     elseif not inmath and item.id == 7 then % Apply recursively
6362         item.replace = item.replace and Babel.locale_map(item.replace)
6363         item.pre      = item.pre and Babel.locale_map(item.pre)
6364         item.post     = item.post and Babel.locale_map(item.post)
6365     elseif item.id == node.id'math' then
6366         inmath = (item.subtype == 0)
6367     end
6368 end
6369 return head
6370 end
6371 }

```

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

```

6372 \newcommand\babelcharproperty[1]{%
6373   \count@=#1\relax
6374   \ifvmode
6375     \expandafter\bbl@chprop
6376   \else
6377     \bbl@error{charproperty-only-vertical}{}{}{}%
6378   \fi}
6379 \newcommand\bbl@chprop[3][\the\count@]{%
6380   \@tempcnta=#1\relax

```

```

6381 \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6382 {\bbl@error{unknown-char-property}}{#2}}}%
6383 {}%
6384 \loop
6385 \bbl@cs{chprop@#2}{#3}%
6386 \ifnum\count@<\@tempcnta
6387 \advance\count@\@ne
6388 \repeat}
6389 %
6390 \def\bbl@chprop@direction#1{%
6391 \directlua{
6392 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6393 Babel.characters[\the\count@]['d'] = '#1'
6394 }}
6395 \let\bbl@chprop@bc\bbl@chprop@direction
6396 %
6397 \def\bbl@chprop@mirror#1{%
6398 \directlua{
6399 Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6400 Babel.characters[\the\count@]['m'] = '\number#1'
6401 }}
6402 \let\bbl@chprop@bmg\bbl@chprop@mirror
6403 %
6404 \def\bbl@chprop@linebreak#1{%
6405 \directlua{
6406 Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6407 Babel.cjk_characters[\the\count@]['c'] = '#1'
6408 }}
6409 \let\bbl@chprop@lb\bbl@chprop@linebreak
6410 %
6411 \def\bbl@chprop@locale#1{%
6412 \directlua{
6413 Babel.chr_to_loc = Babel.chr_to_loc or {}
6414 Babel.chr_to_loc[\the\count@] =
6415 \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
6416 }}

```

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.

```

6417 \directlua{% DL7
6418 Babel.nohyphenation = \the\l@nohyphenation
6419 }

```

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

```

6420 \begingroup
6421 \catcode`\~ =12
6422 \catcode`\% =12
6423 \catcode`\& =14
6424 \catcode`\| =12
6425 \gdef\babelprehyphenation{%&
6426 \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{]}
6427 \gdef\babelposthyphenation{%&
6428 \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}{]}
6429 %
6430 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6431 \ifcase#1

```

```

6432 \bbl@activateprehyphen
6433 \or
6434 \bbl@activateposthyphen
6435 \fi
6436 \begingroup
6437 \def\babeltempa{\bbl@add@list\babeltempb}&%
6438 \let\babeltempb\@empty
6439 \def\bbl@tempa{#5}&%
6440 \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
6441 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6442 \bbl@ifsamestring{##1}{remove}&%
6443 {\bbl@add@list\babeltempb{nil}}&%
6444 {\directlua{
6445 local rep = {[##1]=}
6446 local three_args = '%s*=%s*([%-d%.%a{ }|]+)%s+([%-d%.%a{ }|]+)%s+([%-d%.%a{ }|]+)'
6447 &% Numeric passes directly: kern, penalty...
6448 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6449 rep = rep:gsub('^%s*(insert)%s*', ', 'insert = true, ')
6450 rep = rep:gsub('^%s*(after)%s*', ', 'after = true, ')
6451 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6452 rep = rep:gsub('node%s*=%s*(%a+)%s*(%a*)', Babel.capture_node)
6453 rep = rep:gsub(' (norule)' .. three_args,
6454 'norule = {' .. '%2, %3, %4' .. '})')
6455 if #1 == 0 or #1 == 2 then
6456 rep = rep:gsub(' (space)' .. three_args,
6457 'space = {' .. '%2, %3, %4' .. '})')
6458 rep = rep:gsub(' (spacefactor)' .. three_args,
6459 'spacefactor = {' .. '%2, %3, %4' .. '})')
6460 rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6461 &% Transform values
6462 rep, n = rep:gsub( '{{[^\s%-]+}|([^\s%_]+)}' ,
6463 function(v,d)
6464 return string.format (
6465 '\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6466 v,
6467 load( 'return Babel.locale_props'..
6468 '\the\csname bbl@id@@#3\endcsname].' .. d)() )
6469 end )
6470 rep, n = rep:gsub( '{{[^\s%-]+}|([^\s%-]+)}' ,
6471 '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6472 end
6473 if #1 == 1 then
6474 rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6475 rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6476 rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6477 end
6478 tex.print([\string\babeltempa{ } .. rep .. {}])
6479 }}&%
6480 \bbl@foreach\babeltempb{&%
6481 \bbl@forkv{##1}{&%
6482 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6483 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6484 \ifin@else
6485 \bbl@error{bad-transform-option}{###1}{ }&%
6486 \fi}}&%
6487 \let\bbl@kv@attribute\relax
6488 \let\bbl@kv@label\relax
6489 \let\bbl@kv@fonts\@empty
6490 \let\bbl@kv@prepend\relax
6491 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6492 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6493 \ifx\bbl@kv@attribute\relax
6494 \ifx\bbl@kv@label\relax\else

```

```

6495 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6496 \bbl@replace\bbl@kv@fonts{ }{,}&%
6497 \edef\bbl@kv@attribute{\bbl@ATR\bbl@kv@label @#3\bbl@kv@fonts}&%
6498 \count@z@
6499 \def\bbl@elt##1##2##3{&%
6500 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6501 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6502 {\count@\@ne}&%
6503 {\bbl@error{font-conflict-transforms}{}}{}}&%
6504 {}}&%
6505 \bbl@transfont@list
6506 \ifnum\count@=\z@
6507 \bbl@exp{\global\bbl@add\bbl@transfont@list
6508 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6509 \fi
6510 \bbl@ifunset{\bbl@kv@attribute}&%
6511 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6512 {}&%
6513 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6514 \fi
6515 \else
6516 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6517 \fi
6518 \directlua{
6519 local lbkr = Babel.linebreaking.replacements[#1]
6520 local u = unicode.utf8
6521 local id, attr, label
6522 if #1 == 0 then
6523 id = \the\csname bbl@id@#3\endcsname\space
6524 else
6525 id = \the\csname l@#3\endcsname\space
6526 end
6527 \ifx\bbl@kv@attribute\relax
6528 attr = -1
6529 \else
6530 attr = luatexbase.registernumber'\bbl@kv@attribute'
6531 \fi
6532 \ifx\bbl@kv@label\relax\else &% Same refs:
6533 label = [==[\bbl@kv@label]==]
6534 \fi
6535 &% Convert pattern:
6536 local patt = string.gsub([==[#4]==], '%s', '')
6537 if #1 == 0 then
6538 patt = string.gsub(patt, '|', ' ')
6539 end
6540 if not u.find(patt, '()', nil, true) then
6541 patt = '()' .. patt .. '()'
6542 end
6543 patt = string.gsub(patt, '%(%)^', '^()')
6544 patt = string.gsub(patt, '%$(%)', '()$')
6545 patt = u.gsub(patt, '{(.)}',
6546 function (n)
6547 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6548 end)
6549 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6550 function (n)
6551 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6552 end)
6553 lbkr[id] = lbkr[id] or {}
6554 table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6555 { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6556 }&%
6557 \endgroup}

```

```

6558 \endgroup
6559 %
6560 \let\bbbl@transfont@list\@empty
6561 \def\bbbl@settransfont{%
6562   \global\let\bbbl@settransfont\relax % Execute only once
6563   \gdef\bbbl@transfont{%
6564     \def\bbbl@elt####1####2####3{%
6565       \bbbl@ifblank{####3}%
6566       {\count@tw@}% Do nothing if no fonts
6567       {\count@z@
6568         \bbbl@vforeach{####3}{%
6569           \def\bbbl@tempd{#####1}%
6570           \edef\bbbl@tempe{\bbbl@transfam/\f@series/\f@shape}%
6571           \ifx\bbbl@tempd\bbbl@tempe
6572             \count@\@ne
6573           \else\ifx\bbbl@tempd\bbbl@transfam
6574             \count@\@ne
6575           \fi\fi}%
6576         \ifcase\count@
6577           \bbbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6578         \or
6579           \bbbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6580         \fi}}%
6581     \bbbl@transfont@list}%
6582 \AddToHook{selectfont}{\bbbl@transfont}% Hooks are global.
6583 \gdef\bbbl@transfam{-unknown-}%
6584 \bbbl@foreach\bbbl@font@fams{%
6585   \AddToHook{##1family}{\def\bbbl@transfam{##1}}%
6586   \bbbl@ifsamestring{\@nameuse{##1default}}\familydefault
6587   {\xdef\bbbl@transfam{##1}}%
6588   {}}}
6589 %
6590 \DeclareRobustCommand\enablelocaletransform[1]{%
6591   \bbbl@ifunset{\bbbl@ATR@#1@\language @}%
6592   {\bbbl@error{transform-not-available}{#1}{}}}%
6593   {\bbbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6594 \DeclareRobustCommand\disablelocaletransform[1]{%
6595   \bbbl@ifunset{\bbbl@ATR@#1@\language @}%
6596   {\bbbl@error{transform-not-available-b}{#1}{}}}%
6597   {\bbbl@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`.

```

6598 \def\bbbl@activateposthyphen{%
6599   \let\bbbl@activateposthyphen\relax
6600   \ifx\bbbl@attr@hboxed\@undefined
6601     \newattribute\bbbl@attr@hboxed
6602   \fi
6603   \directlua{
6604     require('babel-transforms.lua')
6605     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6606   }}
6607 \def\bbbl@activateprehyphen{%
6608   \let\bbbl@activateprehyphen\relax
6609   \ifx\bbbl@attr@hboxed\@undefined
6610     \newattribute\bbbl@attr@hboxed
6611   \fi
6612   \directlua{
6613     require('babel-transforms.lua')
6614     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6615   }}
6616 \newcommand\SetTransformValue[3]{%
6617   \directlua{

```

```

6618   Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6619 }}

```

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.

```

6620 \newcommand\ShowBabelTransforms[1]{%
6621   \bbl@activateprehyphen
6622   \bbl@activateposthyphen
6623   \begingroup
6624     \directlua{ Babel.show_transforms = true }%
6625     \setbox\z@\vbox{#1}%
6626     \directlua{ Babel.show_transforms = false }%
6627   \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.

```

6628 \newcommand\localeprehyphenation[1]{%
6629   \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.

```

6630 \def\bbl@activate@preotf{%
6631   \let\bbl@activate@preotf\relax % only once
6632   \directlua{
6633     function Babel.pre_otfload_v(head)
6634       if Babel.numbers and Babel.digits_mapped then
6635         head = Babel.numbers(head)
6636       end
6637       if Babel.bidi_enabled then
6638         head = Babel.bidi(head, false, dir)
6639       end
6640       return head
6641     end
6642     %
6643     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6644       if Babel.numbers and Babel.digits_mapped then
6645         head = Babel.numbers(head)
6646       end
6647       if Babel.bidi_enabled then
6648         head = Babel.bidi(head, false, dir)
6649       end
6650       return head
6651     end
6652     %
6653     luatexbase.add_to_callback('pre_linebreak_filter',
6654       Babel.pre_otfload_v,
6655       'Babel.pre_otfload_v',
6656       Babel.priority_in_callback('pre_linebreak_filter',
6657         'luaotfload.node_processor') or nil)
6658     %
6659     luatexbase.add_to_callback('hpack_filter',
6660       Babel.pre_otfload_h,
6661       'Babel.pre_otfload_h',
6662       Babel.priority_in_callback('hpack_filter',
6663         'luaotfload.node_processor') or nil)
6664   }}

```

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

```

6665 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6666   \let\bbl@beforeforeign\leavevmode
6667   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6668   \RequirePackage{luatexbase}
6669   \bbl@activate@preotf
6670   \directlua{
6671     require('babel-data-bidi.lua')
6672     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6673       require('babel-bidi-basic.lua')
6674     \or
6675       require('babel-bidi-basic-r.lua')
6676       table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6677       table.insert(Babel.ranges, {0xF0000, 0xFFFFD, 'on'})
6678       table.insert(Babel.ranges, {0x100000, 0x10FFFD, 'on'})
6679     \fi}
6680   \newattribute\bbl@attr@dir
6681   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6682   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6683 \fi
6684 %
6685 \chardef\bbl@thetextdir\z@
6686 \chardef\bbl@thepardir\z@
6687 \def\bbl@setluadir#1#2{% 1=\text/pardirection 2= 0:l 1:r 2:a1
6688   \ifcase#2\relax
6689     \ifcase#1\else#1=\z@\fi
6690   \else
6691     \ifcase#1#1=\@ne\fi
6692   \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.

```

6693 \def\bbl@thedir{0}
6694 \def\bbl@textdir#1{%
6695   \bbl@setluadir\textdirection{#1}%
6696   \chardef\bbl@thetextdir#1\relax
6697   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6698   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6699 \def\bbl@pardir#1{% Used twice
6700   \bbl@setluadir\pardirection{#1}%
6701   \chardef\bbl@thepardir#1\relax}
6702 \def\bbl@bodydir{\bbl@setluadir\bodydirection}% Used once
6703 \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.

```

6704 \ifnum\bbl@bidimode>\z@ % Any bidi=
6705   \def\bbl@insidemath{0}%
6706   \def\bbl@everymath{\def\bbl@insidemath{1}}
6707   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6708   \frozen@everymath\expandafter{%
6709     \expandafter\bbl@everymath\the\frozen@everymath}
6710   \frozen@everydisplay\expandafter{%
6711     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6712   \AtBeginDocument{
6713     \directlua{
6714       function Babel.math_box_dir(head)
6715         if not (token.get_macro('bbl@insidemath') == '0') then
6716           if Babel.hlist_has_bidi(head) then
6717             local d = node.new(node.id'dir')

```

```

6718         d.dir = '+TRT'
6719         for item in node.traverse(head) do
6720             if item.id == 11 or item.id == node.id'glyph' then
6721                 head = node.insert_before(head, item, d)
6722                 break
6723             end
6724         end
6725         local inmath = false
6726         for item in node.traverse(head) do
6727             if item.id == 11 then
6728                 inmath = (item.subtype == 0)
6729             elseif not inmath then
6730                 node.set_attribute(item,
6731                     Babel.attr_dir, token.get_macro('bbl@thedir'))
6732             end
6733         end
6734     end
6735 end
6736 return head
6737 end
6738 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6739     "Babel.math_box_dir", 0)
6740 if Babel.unset_atdir then
6741     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6742         "Babel.unset_atdir")
6743     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6744         "Babel.unset_atdir")
6745 end
6746 luatexbase.add_to_callback("post_mlist_to_hlist_filter", function(head)
6747     local dir = tex.mathdirection
6748     local n = node.new("dir")
6749     n.direction = dir
6750     head = node.insert_before(head, head, n)
6751     n = node.new("dir", 1)
6752     n.direction = dir
6753     head = node.insert_after(head, node.tail(head), n)
6754     return head
6755 end, "Babel.ensure_math_dir")
6756 } }%
6757 \fi

```

Experimental. Tentative name.

```

6758 \DeclareRobustCommand\localebox[1]{%
6759     {\def\bbl@insidemath{0}%
6760         \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, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```

6761 \bbl@trace{Redefinitions for bidi layout}
6762 %
6763 <<{*More package options}>> ≡
6764 \chardef\bbl@eqnpos\z@
6765 \DeclareOption{leqno}{\chardef\bbl@eqnpos@ne}
6766 \DeclareOption{fleqn}{\chardef\bbl@eqnpos@tw@}
6767 <</More package options>>
6768 %
6769 \ifnum\bbl@bidimode>\z@ % Any bidi=
6770 \matheqdirmode@ne % Several luatex primitives.
6771 \mathemptydisplaymode@ne % For fixes.
6772 \breakafterdirmode@ne %
6773 \fixupboxesmode@ne %
6774 \let\bbl@eqnudir\relax
6775 \def\bbl@eqdel{()}
6776 \def\bbl@eqnum{%
6777   {\normalfont\normalcolor
6778     \expandafter\@firstoftwo\bbl@eqdel
6779     \theequation
6780     \expandafter\@secondoftwo\bbl@eqdel}}
6781 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6782 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6783 \def\bbl@eqno@flip#1{% So
6784   \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6785     \eqno
6786     \hb@xt@.01pt{%
6787       \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6788   \else
6789     \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6790   \fi
6791   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6792 \def\bbl@leqno@flip#1{%
6793   \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6794     \leqno
6795     \hb@xt@.01pt{%
6796       \hss\hb@xt@\displaywidth{{#1}\glet\bbl@upset\@currentlabel}\hss}}%
6797   \else
6798     \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6799   \fi
6800   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6801 %
6802 \AtBeginDocument{%
6803   \ifx\bbl@noamsmath\relax\else
6804     \ifx\maketag@@@undefined % Normal equation, eqnarray
6805       \ifnum\bbl@eqnpos=\tw@
6806         \bbl@replace\equation{\hb@xt@\linewidth}%
6807           {\hbox bdir\mathdirection to\linewidth}%
6808         \bbl@carg\bbl@sreplace[ ]{\hb@xt@\linewidth}%
6809           {\hbox bdir\mathdirection to\linewidth}%
6810       \fi
6811       \AddToHook{env/equation/begin}{%
6812         \ifnum\bbl@thetextdir>\z@
6813           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6814           \let\@eqnnum\bbl@eqnum
6815           \edef\bbl@eqnudir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6816           \chardef\bbl@thetextdir\z@
6817           \bbl@add\normalfont{\bbl@eqnudir}%
6818           \ifcase\bbl@eqnpos
6819             \let\bbl@puteqno\bbl@eqno@flip

```

```

6820         \or
6821         \let\bbl@puteqno\bbl@leqno@flip
6822         \fi
6823     \fi}%
6824 \ifnum\bbl@eqnpos=\tw@%else
6825 \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6826 \fi
6827 \AddToHook{env/eqnarray/begin}{%
6828     \ifnum\bbl@thetextdir>\z@
6829     \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6830     \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6831     \chardef\bbl@thetextdir\z@
6832     \bbl@add\normalfont{\bbl@eqnodir}%
6833     \ifnum\bbl@eqnpos=\@ne
6834     \def\@eqnnum{%
6835         \setbox\z@\hbox{\bbl@eqnum}%
6836         \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6837     \else
6838     \let\@eqnnum\bbl@eqnum
6839     \fi
6840 \fi}
6841 \else % amstex
6842     \ifnum\bbl@eqnpos=\tw@
6843     \bbl@exp{% Hack to hide maybe undefined conditionals:
6844         \\bbl@sreplace\\multline@crcr{\<@flegn>\tabskip\z@skip\<@fi>\crrc}}%
6845     \fi
6846     \bbl@exp{% Hack to hide maybe undefined conditionals:
6847         \chardef\bbl@eqnpos=0%
6848         \<iftagsleft@>1\<else>\<@flegn>2\<@fi>\<@fi>\relax}%
6849     \ifnum\bbl@eqnpos=\@ne
6850     \let\bbl@ams@lap\hbox
6851     \else
6852     \let\bbl@ams@lap\llap
6853     \fi
6854     \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6855     \bbl@sreplace\intertext@{\normalbaselines}%
6856     {\normalbaselines
6857         \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6858     \ExplSyntaxOff
6859     \def\bbl@ams@tagbox#1#2{\#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6860     \ifx\bbl@ams@lap\hbox % leqno
6861     \def\bbl@ams@flip#1{%
6862         \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6863     \else % eqno
6864     \def\bbl@ams@flip#1{%
6865         \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6866     \fi
6867     \def\bbl@ams@preset#1{%
6868         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6869         \ifnum\bbl@thetextdir>\z@
6870         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6871         \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6872         \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6873         \fi}%
6874     \def\bbl@ams@equation{%
6875         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6876         \ifnum\bbl@thetextdir>\z@
6877         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6878         \chardef\bbl@thetextdir\z@
6879         \bbl@add\normalfont{\bbl@eqnodir}%
6880         \ifcase\bbl@eqnpos
6881         \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6882         \or

```

```

6883         \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6884     \fi
6885 \fi}%
6886 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6887 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6888 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6889 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6890 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6891 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6892 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6893 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6894 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6895 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6896 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6897 % Hackish, for proper alignment. Don't ask me why it works!:
6898 \bbl@exp{% Avoid a 'visible' conditional
6899     \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}%
6900     \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6901 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6902 \AddToHook{env/split/before}{%
6903     \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6904     \ifnum\bbl@thetextdir>z@
6905         \bbl@ifsamestring\@currentvir{equation}%
6906         {\ifx\bbl@ams@lap\hbox % leqno
6907             \def\bbl@ams@flip#1{%
6908                 \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6909             \else
6910                 \def\bbl@ams@flip#1{%
6911                     \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6912                 \fi}%
6913             {}%
6914         \fi}%
6915 \fi\fi}
6916 \fi

```

Declarations specific to lua, called by \babelprovide.

```

6917 \def\bbl@provide@extra#1{%
6918     % == onchar ==
6919     \ifx\bbl@KVP@onchar\@nnil\else
6920         \bbl@luahyphenate
6921         \bbl@exp{%
6922             \\\AddToHook{env/document/before}{%
6923                 {\let\\bbl@ifrestoring\\@firstoftwo
6924                     \\\select@language{#1}}}%
6925             \directlua{
6926                 if Babel.locale_mapped == nil then
6927                     Babel.locale_mapped = true
6928                     Babel.linebreaking.add_before(Babel.locale_map, 1)
6929                     Babel.loc_to_scr = {}
6930                     Babel.chr_to_loc = Babel.chr_to_loc or {}
6931                 end
6932                 Babel.locale_props[\the\localeid].letters = false
6933             }%
6934             \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6935             \ifin@
6936                 \directlua{
6937                     Babel.locale_props[\the\localeid].letters = true
6938                 }%
6939             \fi
6940             \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6941             \ifin@
6942                 \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6943                     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%

```

```

6944 \fi
6945 \bbl@exp{\bbl@add\bbl@starthyphens
6946 {\bbl@patterns@lua{\language}}}%
6947 \directlua{
6948   if Babel.script_blocks['\bbl@cl{sbc}'] then
6949     Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6950     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@language}\space
6951   end
6952 }%
6953 \fi
6954 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6955 \ifin@
6956 \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys@language}}}%
6957 \bbl@ifunset{bbl@wdir@language}{\bbl@provide@dirs@language}}}%
6958 \directlua{
6959   if Babel.script_blocks['\bbl@cl{sbc}'] then
6960     Babel.loc_to_scr[\the\localeid] =
6961       Babel.script_blocks['\bbl@cl{sbc}']
6962   end}%
6963 \ifx\bbl@mapselect\undefined
6964 \AtBeginDocument{%
6965   \bbl@patchfont{\bbl@mapselect}}%
6966   {\selectfont}}%
6967 \def\bbl@mapselect{%
6968   \let\bbl@mapselect\relax
6969   \edef\bbl@prefontid{\fontid\font}}%
6970 \def\bbl@mapdir##1{%
6971   \begingroup
6972     \setbox\z@\hbox{% Force text mode
6973       \def\language{##1}%
6974       \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6975       \bbl@switchfont
6976       \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6977         \directlua{
6978           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6979             [\bbl@prefontid] = \fontid\font\space}%
6980         \fi}%
6981       \endgroup}%
6982 \fi
6983 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir@language}}}%
6984 \fi
6985 \fi
6986 % == mapfont ==
6987 % For bidi texts, to switch the font based on direction. Deprecated
6988 \ifx\bbl@KVP@mapfont\@nnil\else
6989   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{%
6990     {\bbl@error{unknown-mapfont}}}%
6991   \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys@language}}}%
6992   \bbl@ifunset{bbl@wdir@language}{\bbl@provide@dirs@language}}}%
6993 \ifx\bbl@mapselect\undefined
6994 \AtBeginDocument{%
6995   \bbl@patchfont{\bbl@mapselect}}%
6996   {\selectfont}}%
6997 \def\bbl@mapselect{%
6998   \let\bbl@mapselect\relax
6999   \edef\bbl@prefontid{\fontid\font}}%
7000 \def\bbl@mapdir##1{%
7001   {\def\language{##1}%
7002     \let\bbl@ifrestoring\@firstoftwo % avoid font warning
7003     \bbl@switchfont
7004     \directlua{Babel.fontmap
7005       [\the\csname bbl@wdir@##1\endcsname]%
7006       [\bbl@prefontid]=\fontid\font}}}%

```

```

7007 \fi
7008 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
7009 \fi
7010 % == Line breaking: CJK quotes ==
7011 \ifcase\bbl@engine\or
7012 \bbl@xin{/c}{/\bbl@ccl{lnbrk}}%
7013 \ifin@
7014 \bbl@ifunset{\bbl@quote@\language}%
7015 {\directlua{
7016     Babel.locale_props[\the\localeid].cjk_quotes = {}
7017     local cs = 'op'
7018     for c in string.utfvalues(
7019         [[\csname \bbl@quote@\language\endcsname]]) do
7020         if Babel.cjk_characters[c].c == 'qu' then
7021             Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
7022         end
7023         cs = (cs == 'op') and 'cl' or 'op'
7024     end
7025 }}%
7026 \fi
7027 \fi
7028 % == Counters: mapdigits ==
7029 % Native digits
7030 \ifx\bbl@KVP@mapdigits\@nnil\else
7031 \bbl@ifunset{\bbl@dgnat@\language}%
7032 {\bbl@activate@preotf
7033 \directlua{
7034     Babel.digits_mapped = true
7035     Babel.digits = Babel.digits or {}
7036     Babel.digits[\the\localeid] =
7037         table.pack(string.utfvalue('\bbl@ccl{dgnat}'))
7038     if not Babel.numbers then
7039         function Babel.numbers(head)
7040             local LOCALE = Babel.attr_locale
7041             local GLYPH = node.id'glyph'
7042             local inmath = false
7043             for item in node.traverse(head) do
7044                 if not inmath and item.id == GLYPH then
7045                     local temp = node.get_attribute(item, LOCALE)
7046                     if Babel.digits[temp] then
7047                         local chr = item.char
7048                         if chr > 47 and chr < 58 then
7049                             item.char = Babel.digits[temp][chr-47]
7050                         end
7051                     end
7052                 elseif item.id == node.id'math' then
7053                     inmath = (item.subtype == 0)
7054                 end
7055             end
7056             return head
7057         end
7058     end
7059 }}%
7060 \fi
7061 % == transforms ==
7062 \ifx\bbl@KVP@transforms\@nnil\else
7063 \def\bbl@elt##1##2##3{%
7064     \in@{$transforms.}{$##1}%
7065     \ifin@
7066     \def\bbl@tempa{##1}%
7067     \bbl@replace\bbl@tempa{transforms.}%
7068     \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
7069 \fi}%

```

```

7070 \bbl@exp{%
7071   \\\bbl@ifblank{\bbl@cl{dgnat}}}%
7072   {\let\\bbl@tempa\relax}%
7073   {\def\\bbl@tempa{%
7074     \\\bbl@elt{transforms.prehyphenation}%
7075     {digits.native.1.0}{([0-9])}%
7076     \\\bbl@elt{transforms.prehyphenation}%
7077     {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}%
7078 \ifx\bbl@tempa\relax\else
7079   \toks@{\expandafter\expandafter\expandafter{%
7080     \csname bbl@inidata@\language\endcsname}%
7081     \bbl@csarg\edef{inidata@\language}{%
7082       \unexpanded\expandafter{\bbl@tempa}%
7083       \the\toks@}%
7084   \fi
7085   \csname bbl@inidata@\language\endcsname
7086   \bbl@release@transforms\relax % \relax closes the last item.
7087 \fi}

```

Start tabular here:

```

7088 \def\localerestoredirs{%
7089   \ifcase\bbl@thetextdir
7090     \ifnum\textdirection=\z@\else\textdirection=\z@\fi
7091   \else
7092     \ifnum\textdirection=\@ne\else\textdirection=\@ne\fi
7093   \fi
7094   \ifcase\bbl@thepardir
7095     \ifnum\pardirection=\z@\else\pardirection=\z@\bodydirection=\z@\fi
7096   \else
7097     \ifnum\pardirection=\@ne\else\pardirection=\@ne\bodydirection=\@ne\fi
7098   \fi}
7099 %
7100 \IfBabelLayout{tabular}%
7101   {\chardef\bbl@tabular@mode\z@}% All RTL
7102   {\IfBabelLayout{ltrtabular}%
7103     {\chardef\bbl@tabular@mode\@ne}%
7104     {\chardef\bbl@tabular@mode\z@}}% Mixed, with LTR cols
7105 %
7106 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
7107 % Redefine: vrules mess up dirs (why?).
7108 \AtBeginDocument{\def\@arstrut{\relax\copy\@arstrutbox}}%
7109 \ifcase\bbl@tabular@mode\else % 1 = Mixed
7110   \let\bbl@parabefore\relax
7111   \AddToHook{para/before}{\bbl@parabefore}
7112   \AtBeginDocument{%
7113     \bbl@replace@tabular{\hbox\bgroup}{\hbox bdir\z@\bgroup}%
7114     \ifnum\bbl@tabular@mode=\@ne
7115       \bbl@ifunset{@tabclassz}{}%
7116       \bbl@exp{% Hide conditionals
7117         \\\bbl@sreplace\\\@tabclassz
7118         {\<ifcase>\\\@chnum}%
7119         {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
7120     \@ifpackageloaded{colortbl}%
7121     {\bbl@sreplace@classz
7122       {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
7123     {\@ifpackageloaded{array}%
7124       {\bbl@exp{% Hide conditionals
7125         \\\bbl@sreplace\\\@classz
7126         {\<ifcase>\\\@chnum}%
7127         {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
7128         \\\bbl@sreplace\\\@classz
7129         {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
7130       {}}%

```

```

7131 \fi}%
7132 \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.

```

7133 \AtBeginDocument{%
7134 \ifpackageloaded{multicol}%
7135 {\toks@ \expandafter{\multi@column@out}%
7136 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
7137 {}%
7138 \ifpackageloaded{paracol}%
7139 {\edef\pcol@output{%
7140 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7141 {}}%
7142 \fi

```

Finish here if there in no layout.

```

7143 \ifx\bbl@opt@layout\@nnil\endinput\fi

```

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

```

7144 \chardef\bbl@trace@vboxdir\z@
7145 \ifnum\bbl@bidimode>\z@ % Any bidi=
7146 \IfBabelLayout{nopars}{}
7147 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7148 \IfBabelLayout{pars}
7149 {\chardef\bbl@trace@vboxdir\@ne
7150 \def\@hangfrom#1{%
7151 \setbox\@tempboxa\hbox{{#1}}%
7152 \hangindent\wd\@tempboxa
7153 \ifnum\bbl@vbox@bodydir=\pardirection\else
7154 \shapemode\@ne
7155 \fi
7156 \noindent\box\@tempboxa}}
7157 {}
7158 \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).

```

7159 \IfBabelLayout{lists}
7160 {\chardef\bbl@trace@vboxdir\@ne
7161 \let\bbl@OL@list\list
7162 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
7163 \let\bbl@NL@list\list
7164 \def\bbl@listparshape#1#2#3{%
7165 \parshape #1 #2 #3 %
7166 \ifnum\bbl@vbox@bodydir=\pardirection\else
7167 \shapemode\tw@
7168 \fi}}
7169 {}
7170 %
7171 \ifcase\bbl@trace@vboxdir\else
7172 \AtBeginDocument{\chardef\bbl@vbox@bodydir\pagedirection}%
7173 \def\bbl@vbox@lists{\chardef\bbl@vbox@bodydir\bodydirection}%
7174 \let\bbl@bidi@vbox\everyvbox
7175 \@nameuse{newtoks}\everyvbox % \outer in Plain
7176 \everyvbox\expandafter{\the\bbl@bidi@vbox}%
7177 \bbl@bidi@vbox{\bbl@vbox@lists\the\everyvbox}%
7178 \fi
7179 %
7180 \IfBabelLayout{graphics}
7181 {\let\bbl@pictresetdir\relax
7182 \def\bbl@pictsetdir#1{%
7183 \ifcase\bbl@thetextdir

```

```

7184     \let\bbl@pictresetdir\relax
7185 \else
7186     \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7187     \or\textdir TLT
7188     \else\bodydir TLT \textdir TLT
7189     \fi
7190     % \(\text|par)dir required in pgf:
7191     \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7192 \fi}%
7193 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
7194 \directlua{
7195     Babel.get_picture_dir = true
7196     Babel.picture_has_bidi = 0
7197     %
7198     function Babel.picture_dir (head)
7199         if not Babel.get_picture_dir then return head end
7200         if Babel.hlist_has_bidi(head) then
7201             Babel.picture_has_bidi = 1
7202         end
7203         return head
7204     end
7205     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7206         "Babel.picture_dir")
7207 }%
7208 \AddToHook{package/graphics/after}{%
7209     \bbl@exp{%
7210         \\bbl@sreplace\\rotatebox{\hbox{{\string#2}}}
7211         {\hbox bdir\z@{\hbox bdir<ifcase>\bbl@thetextdir\z@<else>\@ne<fi>{\string#2}}}}}%
7212 \AddToHook{package/graphicsx/after}{%
7213     \bbl@exp{%
7214         \\bbl@sreplace\\Grot@box@std{\hbox{{\string#2}}}
7215         {\hbox bdir\z@{\hbox bdir<ifcase>\bbl@thetextdir\z@<else>\@ne<fi>{\string#2}}}}%
7216         \\bbl@sreplace\\Grot@box@kv{\hbox{\string#3}}
7217         {{\hbox bdir\z@}{\hbox bdir<ifcase>\bbl@thetextdir\z@<else>\@ne<fi>{\string#3}}}}}%
7218         \\bbl@sreplace\\Grot@box{\hbox}{\hbox bdir\z@}}}%
7219 \AtBeginDocument{%
7220     \long\def\put(#1,#2)#3{%
7221         \@killglue
7222         % Try:
7223         \ifx\bbl@pictresetdir\relax
7224             \def\bbl@tempc{0}%
7225         \else
7226             \directlua{
7227                 Babel.get_picture_dir = true
7228                 Babel.picture_has_bidi = 0
7229             }%
7230             \setbox\z@\hb@xt@{z@}%
7231             \@defaultunitsset\@tempdimc{#1}\unitlength
7232             \kern\@tempdimc
7233             #3\hss}%
7234             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
7235         \fi
7236         % Do:
7237         \@defaultunitsset\@tempdimc{#2}\unitlength
7238         \raise\@tempdimc\hb@xt@{z@}%
7239         \@defaultunitsset\@tempdimc{#1}\unitlength
7240         \kern\@tempdimc
7241         {\ifnum\bbl@tempc>z@\bbl@pictresetdir\fi#3}\hss}%
7242         \ignorespaces}%
7243     \MakeRobust\put}%
7244 \AtBeginDocument
7245     {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
7246     \ifx\pgfpicture\@undefined\else

```

```

7247 \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
7248 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
7249 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
7250 \fi
7251 \ifx\tikzpicture\undefined\else
7252 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
7253 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
7254 \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
7255 \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
7256 \fi
7257 }}
7258 {}

```

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`.

```

7259 \IfBabelLayout{counters*}%
7260 {\bbl@add\bbl@opt@layout{.counters.}%
7261 \directlua{
7262 \lua{
7263 \lua{
7264 \lua{
7265 \IfBabelLayout{counters*}%
7266 {\let\bbl@latinarabic=\@arabic
7267 \let\bbl@OL@@arabic=\@arabic
7268 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
7269 \ifpackagewith{babel}{bidi=default}%
7270 {\let\bbl@asciroman=\@roman
7271 \let\bbl@OL@@roman=\@roman
7272 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7273 \let\bbl@asciiRoman=\@Roman
7274 \let\bbl@OL@@roman=\@Roman
7275 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7276 \let\bbl@OL@labelenumii\labelenumii
7277 \def\labelenumii{}\theenumii}%
7278 \let\bbl@OL@p@enumiii\p@enumiii
7279 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}

```

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

```

7280 \IfBabelLayout{extras}%
7281 {\bbl@ncarg\let\bbl@OL@underline{underline }%
7282 \bbl@carg\bbl@sreplace{underline }{\hbox}%
7283 {\def\bbl@insidemath{0}\hbox bdir\ifcase\bbl@thetextdir\z@else\@ne\fi}%
7284 \let\bbl@OL@LaTeXe\LaTeXe
7285 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7286 \if b\expandafter\@car\@f@series\@nil\boldmath\fi
7287 \babelsublr{\LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}}%
7288 {}
7289 </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.

```
7290 (*transforms)
7291 Babel.linebreaking.replacements = {}
7292 Babel.linebreaking.replacements[0] = {} -- pre
7293 Babel.linebreaking.replacements[1] = {} -- post
7294
7295 function Babel.tovalue(v)
7296   if type(v) == 'table' then
7297     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7298   else
7299     return v
7300   end
7301 end
7302
7303 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7304
7305 function Babel.set_hboxed(head, gc)
7306   for item in node.traverse(head) do
7307     node.set_attribute(item, Babel.attr_hboxed, 1)
7308   end
7309   return head
7310 end
7311
7312 Babel.fetch_subtext = {}
7313
7314 Babel.ignore_pre_char = function(node)
7315   return (node.lang == Babel.nohyphenation)
7316 end
7317
7318 Babel.show_transforms = false
7319
7320 -- Merging both functions doesn't seem feasible, because there are too
7321 -- many differences.
7322 Babel.fetch_subtext[0] = function(head)
7323   local word_string = ''
7324   local word_nodes = {}
7325   local lang
7326   local item = head
7327   local inmath = false
7328
7329   while item do
7330     if item.id == 11 then
7331       inmath = (item.subtype == 0)
7332     end
7333
7334     if inmath then
7335       -- pass
7336     elseif item.id == 29 then
7337       local locale = node.get_attribute(item, Babel.attr_locale)
7338
7339       if lang == locale or lang == nil then
7340         lang = lang or locale
7341       if Babel.ignore_pre_char(item) then
7342         word_string = word_string .. Babel.us_char
7343       else
7344         if node.has_attribute(item, Babel.attr_hboxed) then
7345           word_string = word_string .. Babel.us_char
7346         else
7347           word_string = word_string .. unicode.utf8.char(item.char)
7348         end
7349       end
7350     end
```

```

7351         end
7352         word_nodes[#word_nodes+1] = item
7353     else
7354         break
7355     end
7356
7357     elseif item.id == 12 and item.subtype == 13 then
7358         if node.has_attribute(item, Babel.attr_hboxed) then
7359             word_string = word_string .. Babel.us_char
7360         else
7361             word_string = word_string .. ' '
7362         end
7363         word_nodes[#word_nodes+1] = item
7364
7365         -- Ignore leading unrecognized nodes, too.
7366         elseif word_string ~= '' then
7367             word_string = word_string .. Babel.us_char
7368             word_nodes[#word_nodes+1] = item -- Will be ignored
7369         end
7370
7371         item = item.next
7372     end
7373
7374     -- Here and above we remove some trailing chars but not the
7375     -- corresponding nodes. But they aren't accessed.
7376     if word_string:sub(-1) == ' ' then
7377         word_string = word_string:sub(1,-2)
7378     end
7379     if Babel.show_transforms then texio.write_nl(word_string) end
7380     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7381     return word_string, word_nodes, item, lang
7382 end
7383
7384 Babel.fetch_subtext[1] = function(head)
7385     local word_string = ''
7386     local word_nodes = {}
7387     local lang
7388     local item = head
7389     local inmath = false
7390
7391     while item do
7392
7393         if item.id == 11 then
7394             inmath = (item.subtype == 0)
7395         end
7396
7397         if inmath then
7398             -- pass
7399
7400         elseif item.id == 29 then
7401             if item.lang == lang or lang == nil then
7402                 lang = lang or item.lang
7403                 if node.has_attribute(item, Babel.attr_hboxed) then
7404                     word_string = word_string .. Babel.us_char
7405                 elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7406                     word_string = word_string .. Babel.us_char
7407                 else
7408                     word_string = word_string .. unicode.utf8.char(item.char)
7409                 end
7410                 word_nodes[#word_nodes+1] = item
7411             else
7412                 break
7413             end

```

```

7414
7415 elseif item.id == 7 and item.subtype == 2 then
7416     if node.has_attribute(item, Babel.attr_hboxed) then
7417         word_string = word_string .. Babel.us_char
7418     else
7419         word_string = word_string .. '='
7420     end
7421     word_nodes[#word_nodes+1] = item
7422
7423 elseif item.id == 7 and item.subtype == 3 then
7424     if node.has_attribute(item, Babel.attr_hboxed) then
7425         word_string = word_string .. Babel.us_char
7426     else
7427         word_string = word_string .. '|'
7428     end
7429     word_nodes[#word_nodes+1] = item
7430
7431 -- (1) Go to next word if nothing was found, and (2) implicitly
7432 -- remove leading USs.
7433 elseif word_string == '' then
7434     -- pass
7435
7436 -- This is the responsible for splitting by words.
7437 elseif (item.id == 12 and item.subtype == 13) then
7438     break
7439
7440 else
7441     word_string = word_string .. Babel.us_char
7442     word_nodes[#word_nodes+1] = item -- Will be ignored
7443 end
7444
7445 item = item.next
7446 end
7447 if Babel.show_transforms then texio.write_nl(word_string) end
7448 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7449 return word_string, word_nodes, item, lang
7450 end
7451
7452 function Babel.pre_hyphenate_replace(head)
7453     Babel.hyphenate_replace(head, 0)
7454 end
7455
7456 function Babel.post_hyphenate_replace(head)
7457     Babel.hyphenate_replace(head, 1)
7458 end
7459
7460 Babel.us_char = string.char(31)
7461
7462 function Babel.hyphenate_replace(head, mode)
7463     local u = unicode.utf8
7464     local lbkr = Babel.linebreaking.replacements[mode]
7465     local tovalue = Babel.tovalue
7466
7467     local word_head = head
7468
7469     if Babel.show_transforms then
7470         texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7471     end
7472
7473     while true do -- for each subtext block
7474
7475         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7476

```

```

7477 if Babel.debug then
7478     print()
7479     print((mode == 0) and '@@@<' or '@@@>', w)
7480 end
7481
7482 if nw == nil and w == '' then break end
7483
7484 if not lang then goto next end
7485 if not lbkr[lang] then goto next end
7486
7487 -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7488 -- loops are nested.
7489 for k=1, #lbkr[lang] do
7490     local p = lbkr[lang][k].pattern
7491     local r = lbkr[lang][k].replace
7492     local attr = lbkr[lang][k].attr or -1
7493
7494     if Babel.debug then
7495         print('*****', p, mode)
7496     end
7497
7498     -- This variable is set in some cases below to the first *byte*
7499     -- after the match, either as found by u.match (faster) or the
7500     -- computed position based on sc if w has changed.
7501     local last_match = 0
7502     local step = 0
7503
7504     -- For every match.
7505     while true do
7506         if Babel.debug then
7507             print('====')
7508         end
7509         local new -- used when inserting and removing nodes
7510         local dummy_node -- used by after
7511
7512         local matches = { u.match(w, p, last_match) }
7513
7514         if #matches < 2 then break end
7515
7516         -- Get and remove empty captures (with ()'s, which return a
7517         -- number with the position), and keep actual captures
7518         -- (from (...)), if any, in matches.
7519         local first = table.remove(matches, 1)
7520         local last = table.remove(matches, #matches)
7521         -- Non re-fetched substrings may contain \31, which separates
7522         -- subsubstrings.
7523         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7524
7525         local save_last = last -- with A()BC()D, points to D
7526
7527         -- Fix offsets, from bytes to unicode. Explained above.
7528         first = u.len(w:sub(1, first-1)) + 1
7529         last = u.len(w:sub(1, last-1)) -- now last points to C
7530
7531         -- This loop stores in a small table the nodes
7532         -- corresponding to the pattern. Used by 'data' to provide a
7533         -- predictable behavior with 'insert' (w_nodes is modified on
7534         -- the fly), and also access to 'remove'd nodes.
7535         local sc = first-1 -- Used below, too
7536         local data_nodes = {}
7537
7538         local enabled = true
7539         for q = 1, last-first+1 do

```

```

7540     data_nodes[q] = w_nodes[sc+q]
7541     if enabled
7542         and attr > -1
7543         and not node.has_attribute(data_nodes[q], attr)
7544     then
7545         enabled = false
7546     end
7547 end
7548
7549 -- This loop traverses the matched substring and takes the
7550 -- corresponding action stored in the replacement list.
7551 -- sc = the position in substr nodes / string
7552 -- rc = the replacement table index
7553 local rc = 0
7554
7555 ----- TODO. dummy_node?
7556 while rc < last-first+1 or dummy_node do -- for each replacement
7557     if Babel.debug then
7558         print('.....', rc + 1)
7559     end
7560     sc = sc + 1
7561     rc = rc + 1
7562
7563     if Babel.debug then
7564         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7565         local ss = ''
7566         for itt in node.traverse(head) do
7567             if itt.id == 29 then
7568                 ss = ss .. unicode.utf8.char(itt.char)
7569             else
7570                 ss = ss .. '{' .. itt.id .. '}'
7571             end
7572         end
7573         print('*****', ss)
7574     end
7575
7576     local crep = r[rc]
7577     local item = w_nodes[sc]
7578     local item_base = item
7579     local placeholder = Babel.us_char
7580     local d
7581
7582     if crep and crep.data then
7583         item_base = data_nodes[crep.data]
7584     end
7585
7586     if crep then
7587         step = crep.step or step
7588     end
7589
7590     if crep and crep.after then
7591         crep.insert = true
7592         if dummy_node then
7593             item = dummy_node
7594         else -- TODO. if there is a node after?
7595             d = node.copy(item_base)
7596             head, item = node.insert_after(head, item, d)
7597             dummy_node = item
7598         end
7599     end
7600
7601     if crep and not crep.after and dummy_node then

```

```

7603         node.remove(head, dummy_node)
7604         dummy_node = nil
7605     end
7606
7607     if not enabled then
7608         last_match = save_last
7609         goto next
7610
7611     elseif crep and next(crep) == nil then -- = {}
7612         if step == 0 then
7613             last_match = save_last    -- Optimization
7614         else
7615             last_match = utf8.offset(w, sc+step)
7616         end
7617         goto next
7618
7619     elseif crep == nil or crep.remove then
7620         node.remove(head, item)
7621         table.remove(w_nodes, sc)
7622         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7623         sc = sc - 1 -- Nothing has been inserted.
7624         last_match = utf8.offset(w, sc+1+step)
7625         goto next
7626
7627     elseif crep and crep.kashida then
7628         node.set_attribute(item,
7629             Babel.attr_kashida,
7630             crep.kashida)
7631         last_match = utf8.offset(w, sc+1+step)
7632         goto next
7633
7634     elseif crep and crep.string then
7635         local str = crep.string(matches)
7636         if str == '' then -- Gather with nil
7637             node.remove(head, item)
7638             table.remove(w_nodes, sc)
7639             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7640             sc = sc - 1 -- Nothing has been inserted.
7641         else
7642             local loop_first = true
7643             for s in string.utfvalues(str) do
7644                 d = node.copy(item_base)
7645                 d.char = s
7646                 if loop_first then
7647                     loop_first = false
7648                     head, new = node.insert_before(head, item, d)
7649                     if sc == 1 then
7650                         word_head = head
7651                     end
7652                     w_nodes[sc] = d
7653                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7654                 else
7655                     sc = sc + 1
7656                     head, new = node.insert_before(head, item, d)
7657                     table.insert(w_nodes, sc, new)
7658                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7659                 end
7660                 if Babel.debug then
7661                     print('.....', 'str')
7662                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7663                 end
7664             end -- for
7665             node.remove(head, item)

```

```

7666         end -- if ''
7667         last_match = utf8.offset(w, sc+1+step)
7668         goto next
7669
7670     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7671         d = node.new(7, 3) -- (disc, regular)
7672         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7673         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7674         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7675         d.attr = item_base.attr
7676         if crep.pre == nil then -- TeXbook p96
7677             d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7678         else
7679             d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7680         end
7681         placeholder = '|'
7682         head, new = node.insert_before(head, item, d)
7683
7684     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7685         -- ERROR
7686
7687     elseif crep and crep.penalty then
7688         d = node.new(14, 0) -- (penalty, userpenalty)
7689         d.attr = item_base.attr
7690         d.penalty = tovalue(crep.penalty)
7691         head, new = node.insert_before(head, item, d)
7692
7693     elseif crep and crep.space then
7694         -- 655360 = 10 pt = 10 * 65536 sp
7695         d = node.new(12, 13) -- (glue, spaceskip)
7696         local quad = font.getfont(item_base.font).size or 655360
7697         node.setglue(d, tovalue(crep.space[1]) * quad,
7698                       tovalue(crep.space[2]) * quad,
7699                       tovalue(crep.space[3]) * quad)
7700         if mode == 0 then
7701             placeholder = ' '
7702         end
7703         head, new = node.insert_before(head, item, d)
7704
7705     elseif crep and crep.norule then
7706         -- 655360 = 10 pt = 10 * 65536 sp
7707         d = node.new(2, 3) -- (rule, empty) = \no*rule
7708         local quad = font.getfont(item_base.font).size or 655360
7709         d.width = tovalue(crep.norule[1]) * quad
7710         d.height = tovalue(crep.norule[2]) * quad
7711         d.depth = tovalue(crep.norule[3]) * quad
7712         head, new = node.insert_before(head, item, d)
7713
7714     elseif crep and crep.spacefactor then
7715         d = node.new(12, 13) -- (glue, spaceskip)
7716         local base_font = font.getfont(item_base.font)
7717         node.setglue(d,
7718                     tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7719                     tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7720                     tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7721         if mode == 0 then
7722             placeholder = ' '
7723         end
7724         head, new = node.insert_before(head, item, d)
7725
7726     elseif mode == 0 and crep and crep.space then
7727         -- ERROR
7728

```

```

7729     elseif crep and crep.kern then
7730         d = node.new(13, 1)      -- (kern, user)
7731         local quad = font.getfont(item_base.font).size or 655360
7732         d.attr = item_base.attr
7733         d.kern = tovalue(crep.kern) * quad
7734         head, new = node.insert_before(head, item, d)
7735
7736     elseif crep and crep.node then
7737         d = node.new(crep.node[1], crep.node[2])
7738         d.attr = item_base.attr
7739         head, new = node.insert_before(head, item, d)
7740
7741     end -- i.e., replacement cases
7742
7743     -- Shared by disc, space(factor), kern, node and penalty.
7744     if sc == 1 then
7745         word_head = head
7746     end
7747     if crep.insert then
7748         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7749         table.insert(w_nodes, sc, new)
7750         last = last + 1
7751     else
7752         w_nodes[sc] = d
7753         node.remove(head, item)
7754         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7755     end
7756
7757     last_match = utf8.offset(w, sc+1+step)
7758
7759     ::next::
7760
7761     end -- for each replacement
7762
7763     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7764     if Babel.debug then
7765         print('.....', '/')
7766         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7767     end
7768
7769     if dummy_node then
7770         node.remove(head, dummy_node)
7771         dummy_node = nil
7772     end
7773
7774     end -- for match
7775
7776     end -- for patterns
7777
7778     ::next::
7779     word_head = nw
7780 end -- for substring
7781
7782 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7783 return head
7784 end
7785
7786 -- This table stores capture maps, numbered consecutively
7787 Babel.capture_maps = {}
7788
7789 function Babel.esc_hex_to_char(h)
7790     if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7791         tex.getcatcode(tonumber(h, 16)) ~= 12 then

```

```

7792     return string.format([[Uchar"%X ]], tonumber(h,16))
7793 else
7794     return unicode.utf8.char(tonumber(h, 16))
7795 end
7796 end
7797
7798 -- The following functions belong to the next macro
7799 function Babel.capture_func(key, cap)
7800     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
7801     local cnt
7802     local u = unicode.utf8
7803     ret = u.gsub(ret, '{{(%%x%%x%%x+}}', '\x01%1\x04')
7804     ret, cnt = ret:gsub('{{([0-9])|([^\+]|(-))}}', Babel.capture_func_map)
7805     ret = u.gsub(ret, '\x01(%%x%%x%%x+)\x04', Babel.esc_hex_to_char)
7806     ret = ret:gsub("%[%[%]%%.%", '')
7807     ret = ret:gsub("%.%.%[%[%]%%", '')
7808     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7809 end
7810
7811 function Babel.capt_map(from, mapno)
7812     return Babel.capture_maps[mapno][from] or from
7813 end
7814
7815 -- Handle the {n|abc|ABC} syntax in captures
7816 function Babel.capture_func_map(capno, from, to)
7817     local u = unicode.utf8
7818     from = u.gsub(from, '\x01(%%x%%x%%x+)\x04',
7819         function (n)
7820             return u.char(tonumber(n, 16))
7821         end)
7822     to = u.gsub(to, '\x01(%%x%%x%%x+)\x04',
7823         function (n)
7824             return u.char(tonumber(n, 16))
7825         end)
7826     local froms = {}
7827     for s in string.utfcharacters(from) do
7828         table.insert(froms, s)
7829     end
7830     local cnt = 1
7831     table.insert(Babel.capture_maps, {})
7832     local mlen = table.getn(Babel.capture_maps)
7833     for s in string.utfcharacters(to) do
7834         Babel.capture_maps[mlen][froms[cnt]] = s
7835         cnt = cnt + 1
7836     end
7837     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7838         (mlen) .. ")]" .. " ["
7839 end
7840
7841 -- Create/Extend reversed sorted list of kashida weights:
7842 function Babel.capture_kashida(key, wt)
7843     wt = tonumber(wt)
7844     if Babel.kashida_wts then
7845         for p, q in ipairs(Babel.kashida_wts) do
7846             if wt == q then
7847                 break
7848             elseif wt > q then
7849                 table.insert(Babel.kashida_wts, p, wt)
7850                 break
7851             elseif table.getn(Babel.kashida_wts) == p then
7852                 table.insert(Babel.kashida_wts, wt)
7853             end
7854         end
7855     end

```

```

7855 else
7856     Babel.kashida_wts = { wt }
7857 end
7858 return 'kashida = ' .. wt
7859 end
7860
7861 function Babel.capture_node(id, subtype)
7862     local sbt = 0
7863     for k, v in pairs(node.subtypes(id)) do
7864         if v == subtype then sbt = k end
7865     end
7866     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7867 end
7868
7869 -- Experimental: applies prehyphenation transforms to a string (letters
7870 -- and spaces).
7871 function Babel.string_prehyphenation(str, locale)
7872     local n, head, last, res
7873     head = node.new(8, 0) -- dummy (hack just to start)
7874     last = head
7875     for s in string.utfvalues(str) do
7876         if s == 20 then
7877             n = node.new(12, 0)
7878         else
7879             n = node.new(29, 0)
7880             n.char = s
7881         end
7882         node.set_attribute(n, Babel.attr_locale, locale)
7883         last.next = n
7884         last = n
7885     end
7886     head = Babel.hyphenate_replace(head, 0)
7887     res = ''
7888     for n in node.traverse(head) do
7889         if n.id == 12 then
7890             res = res .. ' '
7891         elseif n.id == 29 then
7892             res = res .. unicode.utf8.char(n.char)
7893         end
7894     end
7895     tex.print(res)
7896 end
7897 </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.

```
7898 (*basic-r)
7899 Babel.bidi_enabled = true
7900
7901 require('babel-data-bidi.lua')
7902
7903 local characters = Babel.characters
7904 local ranges = Babel.ranges
7905
7906 local DIR = node.id("dir")
7907
7908 local function dir_mark(head, from, to, outer)
7909   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7910   local d = node.new(DIR)
7911   d.dir = '+' .. dir
7912   node.insert_before(head, from, d)
7913   d = node.new(DIR)
7914   d.dir = '-' .. dir
7915   node.insert_after(head, to, d)
7916 end
7917
7918 function Babel.bidi(head, ispar)
7919   local first_n, last_n          -- first and last char with nums
7920   local last_es                  -- an auxiliary 'last' used with nums
7921   local first_d, last_d          -- first and last char in L/R block
7922   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):

```
7923   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7924   local strong_lr = (strong == 'l') and 'l' or 'r'
7925   local outer = strong
7926
7927   local new_dir = false
7928   local first_dir = false
7929   local inmath = false
7930
7931   local last_lr
7932
7933   local type_n = ''
7934
7935   for item in node.traverse(head) do
7936
```

```

7937 -- three cases: glyph, dir, otherwise
7938 if item.id == node.id'glyph'
7939   or (item.id == 7 and item.subtype == 2) then
7940
7941   local itemchar
7942   if item.id == 7 and item.subtype == 2 then
7943     itemchar = item.replace.char
7944   else
7945     itemchar = item.char
7946   end
7947   local chardata = characters[itemchar]
7948   dir = chardata and chardata.d or nil
7949   if not dir then
7950     for nn, et in ipairs(ranges) do
7951       if itemchar < et[1] then
7952         break
7953       elseif itemchar <= et[2] then
7954         dir = et[3]
7955         break
7956       end
7957     end
7958   end
7959   dir = dir or 'l'
7960   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).

```

7961   if new_dir then
7962     attr_dir = 0
7963     for at in node.traverse(item.attr) do
7964       if at.number == Babel.attr_dir then
7965         attr_dir = at.value & 0x3
7966       end
7967     end
7968     if attr_dir == 1 then
7969       strong = 'r'
7970     elseif attr_dir == 2 then
7971       strong = 'al'
7972     else
7973       strong = 'l'
7974     end
7975     strong_lr = (strong == 'l') and 'l' or 'r'
7976     outer = strong_lr
7977     new_dir = false
7978   end
7979
7980   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7981   dir_real = dir -- We need dir_real to set strong below
7982   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:

```

7983   if strong == 'al' then
7984     if dir == 'en' then dir = 'an' end -- W2
7985     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7986     strong_lr = 'r' -- W3
7987   end

```

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

```

7988 elseif item.id == node.id'dir' and not inmath then
7989     new_dir = true
7990     dir = nil
7991 elseif item.id == node.id'math' then
7992     inmath = (item.subtype == 0)
7993 else
7994     dir = nil          -- Not a char
7995 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>.

```

7996 if dir == 'en' or dir == 'an' or dir == 'et' then
7997     if dir ~= 'et' then
7998         type_n = dir
7999     end
8000     first_n = first_n or item
8001     last_n = last_es or item
8002     last_es = nil
8003 elseif dir == 'es' and last_n then -- W3+W6
8004     last_es = item
8005 elseif dir == 'cs' then          -- it's right - do nothing
8006 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
8007     if strong_lr == 'r' and type_n ~= '' then
8008         dir_mark(head, first_n, last_n, 'r')
8009     elseif strong_lr == 'l' and first_d and type_n == 'an' then
8010         dir_mark(head, first_n, last_n, 'r')
8011         dir_mark(head, first_d, last_d, outer)
8012         first_d, last_d = nil, nil
8013     elseif strong_lr == 'l' and type_n ~= '' then
8014         last_d = last_n
8015     end
8016     type_n = ''
8017     first_n, last_n = nil, nil
8018 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:

```

8019 if dir == 'l' or dir == 'r' then
8020     if dir ~= outer then
8021         first_d = first_d or item
8022         last_d = item
8023     elseif first_d and dir ~= strong_lr then
8024         dir_mark(head, first_d, last_d, outer)
8025         first_d, last_d = nil, nil
8026     end
8027 end

```

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

```

8028 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
8029     item.char = characters[item.char] and
8030         characters[item.char].m or item.char
8031 elseif (dir or new_dir) and last_lr ~= item then
8032     local mir = outer .. strong_lr .. (dir or outer)
8033     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
8034         for ch in node.traverse(node.next(last_lr)) do

```

```

8035         if ch == item then break end
8036         if ch.id == node.id'glyph' and characters[ch.char] then
8037             ch.char = characters[ch.char].m or ch.char
8038         end
8039     end
8040 end
8041 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).

```

8042     if dir == 'l' or dir == 'r' then
8043         last_lr = item
8044         strong = dir_real          -- Don't search back - best save now
8045         strong_lr = (strong == 'l') and 'l' or 'r'
8046     elseif new_dir then
8047         last_lr = nil
8048     end
8049 end

```

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

```

8050     if last_lr and outer == 'r' then
8051         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
8052             if characters[ch.char] then
8053                 ch.char = characters[ch.char].m or ch.char
8054             end
8055         end
8056     end
8057     if first_n then
8058         dir_mark(head, first_n, last_n, outer)
8059     end
8060     if first_d then
8061         dir_mark(head, first_d, last_d, outer)
8062     end

```

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

```

8063     return node.prev(head) or head
8064 end
8065 </basic-r>

```

And here the Lua code for bidi=basic:

```

8066 (*basic)
8067 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
8068
8069 Babel.fontmap = Babel.fontmap or {}
8070 Babel.fontmap[0] = {}          -- l
8071 Babel.fontmap[1] = {}          -- r
8072 Babel.fontmap[2] = {}          -- al/an
8073
8074 -- To cancel mirroring. Also OML, OMS, U?
8075 Babel.symbol_fonts = Babel.symbol_fonts or {}
8076 Babel.symbol_fonts[font.id('tenln')] = true
8077 Babel.symbol_fonts[font.id('tenlnw')] = true
8078 Babel.symbol_fonts[font.id('tencirc')] = true
8079 Babel.symbol_fonts[font.id('tencircw')] = true
8080
8081 Babel.bidi_enabled = true
8082 Babel.mirroring_enabled = true
8083
8084 require('babel-data-bidi.lua')
8085
8086 local characters = Babel.characters
8087 local ranges = Babel.ranges
8088

```

```

8089 local DIR = node.id('dir')
8090 local GLYPH = node.id('glyph')
8091
8092 local function insert_implicit(head, state, outer)
8093   local new_state = state
8094   if state.sim and state.eim and state.sim ~= state.eim then
8095     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
8096     local d = node.new(DIR)
8097     d.dir = '+' .. dir
8098     node.insert_before(head, state.sim, d)
8099     local d = node.new(DIR)
8100     d.dir = '-' .. dir
8101     node.insert_after(head, state.eim, d)
8102   end
8103   new_state.sim, new_state.eim = nil, nil
8104   return head, new_state
8105 end
8106
8107 local function insert_numeric(head, state)
8108   local new
8109   local new_state = state
8110   if state.san and state.ean and state.san ~= state.ean then
8111     local d = node.new(DIR)
8112     d.dir = '+TLT'
8113     _, new = node.insert_before(head, state.san, d)
8114     if state.san == state.sim then state.sim = new end
8115     local d = node.new(DIR)
8116     d.dir = '-TLT'
8117     _, new = node.insert_after(head, state.ean, d)
8118     if state.ean == state.eim then state.eim = new end
8119   end
8120   new_state.san, new_state.ean = nil, nil
8121   return head, new_state
8122 end
8123
8124 local function glyph_not_symbol_font(node)
8125   if node.id == GLYPH then
8126     return not Babel.symbol_fonts[node.font]
8127   else
8128     return false
8129   end
8130 end
8131
8132 -- TODO - \hbox with an explicit dir can lead to wrong results
8133 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
8134 -- was made to improve the situation, but the problem is the 3-dir
8135 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8136 -- well.
8137
8138 function Babel.bidi(head, ispar, hdir)
8139   local d -- d is used mainly for computations in a loop
8140   local prev_d = ''
8141   local new_d = false
8142
8143   local nodes = {}
8144   local outer_first = nil
8145   local inmath = false
8146
8147   local glue_d = nil
8148   local glue_i = nil
8149
8150   local has_en = false
8151   local first_et = nil

```

```

8152
8153 local has_hyperlink = false
8154
8155 local ATDIR = Babel.attr_dir
8156 local attr_d, temp
8157 local locale_d
8158
8159 local save_outer
8160 local locale_d = node.get_attribute(head, ATDIR)
8161 if locale_d then
8162     locale_d = locale_d & 0x3
8163     save_outer = (locale_d == 0 and 'l') or
8164                 (locale_d == 1 and 'r') or
8165                 (locale_d == 2 and 'al')
8166 elseif ispar then -- Or error? Shouldn't happen
8167     -- when the callback is called, we are just _after_ the box,
8168     -- and the textdir is that of the surrounding text
8169     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8170 else -- Empty box
8171     save_outer = ('TRT' == hdir) and 'r' or 'l'
8172 end
8173 local outer = save_outer
8174 local last = outer
8175 -- 'al' is only taken into account in the first, current loop
8176 if save_outer == 'al' then save_outer = 'r' end
8177
8178 local fontmap = Babel.fontmap
8179
8180 for item in node.traverse(head) do
8181
8182     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8183     locale_d = node.get_attribute(item, ATDIR)
8184     node.set_attribute(item, ATDIR, 0x80)
8185
8186     -- In what follows, #node is the last (previous) node, because the
8187     -- current one is not added until we start processing the neutrals.
8188     -- three cases: glyph, dir, otherwise
8189     if glyph_not_symbol_font(item)
8190         or (item.id == 7 and item.subtype == 2) then
8191
8192         if inmath then goto nextnode end
8193
8194         if locale_d == 0x80 then goto nextnode end
8195         locale_d = locale_d or ((save_outer=='l') and 0 or 1)
8196
8197         local d_font = nil
8198         local item_r
8199         if item.id == 7 and item.subtype == 2 then
8200             item_r = item.replace -- automatic discs have just 1 glyph
8201         else
8202             item_r = item
8203         end
8204
8205         local chardata = characters[item_r.char]
8206         d = chardata and chardata.d or nil
8207         if not d or d == 'nsm' then
8208             for nn, et in ipairs(ranges) do
8209                 if item_r.char < et[1] then
8210                     break
8211                 elseif item_r.char <= et[2] then
8212                     if not d then d = et[3]
8213                     elseif d == 'nsm' then d_font = et[3]
8214                     end

```

```

8215         break
8216     end
8217 end
8218 end
8219 d = d or 'l'
8220
8221 -- A short 'pause' in bidi for mapfont
8222 -- %%% TODO. move if fontmap here
8223 d_font = d_font or d
8224 d_font = (d_font == 'l' and 0) or
8225           (d_font == 'nsm' and 0) or
8226           (d_font == 'r' and 1) or
8227           (d_font == 'al' and 2) or
8228           (d_font == 'an' and 2) or nil
8229 if d_font and fontmap and fontmap[d_font][item_r.font] then
8230     item_r.font = fontmap[d_font][item_r.font]
8231 end
8232
8233 if new_d then
8234     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8235     if inmath then
8236         attr_d = 0
8237     else
8238         attr_d = locale_d & 0x3
8239     end
8240     if attr_d == 1 then
8241         outer_first = 'r'
8242         last = 'r'
8243     elseif attr_d == 2 then
8244         outer_first = 'r'
8245         last = 'al'
8246     else
8247         outer_first = 'l'
8248         last = 'l'
8249     end
8250     outer = last
8251     has_en = false
8252     first_et = nil
8253     new_d = false
8254 end
8255
8256 if glue_d then
8257     if (d == 'l' and 'l' or 'r') ~= glue_d then
8258         table.insert(nodes, {glue_i, 'on', nil})
8259     end
8260     glue_d = nil
8261     glue_i = nil
8262 end
8263
8264 elseif item.id == DIR then
8265     if inmath then goto nextnode end
8266     d = nil
8267     new_d = true
8268
8269 elseif item.id == node.id'glue' and item.subtype == 13 then
8270     if inmath then goto nextnode end
8271     glue_d = d
8272     glue_i = item
8273     d = nil
8274
8275 elseif item.id == node.id'math' then
8276     if item.subtype == 0 then -- mathon
8277         inmath = true

```

```

8278         d = 'on'
8279         table.insert(nodes, {item, 'on', outer_first})
8280         outer_first = nil
8281         goto nextnode
8282     else -- mathoff
8283         inmath = false
8284     end
8285
8286     elseif item.id == 8 and item.subtype == 19 then
8287         has_hyperlink = true
8288
8289     else
8290         d = nil
8291     end
8292
8293     -- AL <= EN/ET/ES      -- W2 + W3 + W6
8294     if last == 'al' and d == 'en' then
8295         d = 'an'          -- W3
8296     elseif last == 'al' and (d == 'et' or d == 'es') then
8297         d = 'on'          -- W6
8298     end
8299
8300     -- EN + CS/ES + EN      -- W4
8301     if d == 'en' and #nodes >= 2 then
8302         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8303             and nodes[#nodes-1][2] == 'en' then
8304             nodes[#nodes][2] = 'en'
8305         end
8306     end
8307
8308     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8309     if d == 'an' and #nodes >= 2 then
8310         if (nodes[#nodes][2] == 'cs')
8311             and nodes[#nodes-1][2] == 'an' then
8312             nodes[#nodes][2] = 'an'
8313         end
8314     end
8315
8316     -- ET/EN                -- W5 + W7->l / W6->on
8317     if d == 'et' then
8318         first_et = first_et or (#nodes + 1)
8319     elseif d == 'en' then
8320         has_en = true
8321         first_et = first_et or (#nodes + 1)
8322     elseif first_et then -- d may be nil here !
8323         if has_en then
8324             if last == 'l' then
8325                 temp = 'l' -- W7
8326             else
8327                 temp = 'en' -- W5
8328             end
8329         else
8330             temp = 'on' -- W6
8331         end
8332         for e = first_et, #nodes do
8333             if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8334         end
8335         first_et = nil
8336         has_en = false
8337     end
8338
8339     -- Force mathdir in math if ON (currently works as expected only
8340     -- with 'l')

```

```

8341
8342   if inmath and d == 'on' then
8343       d = ('TRT' == tex.mathdir) and 'r' or 'l'
8344   end
8345
8346   if d then
8347       if d == 'al' then
8348           d = 'r'
8349           last = 'al'
8350       elseif d == 'l' or d == 'r' then
8351           last = d
8352       end
8353       prev_d = d
8354       table.insert(nodes, {item, d, outer_first})
8355   end
8356
8357   outer_first = nil
8358
8359   ::nextnode::
8360
8361 end -- for each node
8362
8363 -- TODO -- repeated here in case EN/ET is the last node. Find a
8364 -- better way of doing things:
8365 if first_et then      -- dir may be nil here !
8366     if has_en then
8367         if last == 'l' then
8368             temp = 'l'    -- W7
8369         else
8370             temp = 'en'    -- W5
8371         end
8372     else
8373         temp = 'on'    -- W6
8374     end
8375     for e = first_et, #nodes do
8376         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8377     end
8378 end
8379
8380 -- dummy node, to close things
8381 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8382
8383 ----- NEUTRAL -----
8384
8385 outer = save_outer
8386 last = outer
8387
8388 local first_on = nil
8389
8390 for q = 1, #nodes do
8391     local item
8392
8393     local outer_first = nodes[q][3]
8394     outer = outer_first or outer
8395     last = outer_first or last
8396
8397     local d = nodes[q][2]
8398     if d == 'an' or d == 'en' then d = 'r' end
8399     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8400
8401     if d == 'on' then
8402         first_on = first_on or q
8403     elseif first_on then

```

```

8404     if last == d then
8405         temp = d
8406     else
8407         temp = outer
8408     end
8409     for r = first_on, q - 1 do
8410         nodes[r][2] = temp
8411         item = nodes[r][1]    -- MIRRORING
8412         if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8413             and temp == 'r' and characters[item.char] then
8414             local font_mode = ''
8415             if item.font > 0 and font.fonts[item.font].properties then
8416                 font_mode = font.fonts[item.font].properties.mode
8417             end
8418             if font_mode ~= 'harf' and font_mode ~= 'plug' then
8419                 item.char = characters[item.char].m or item.char
8420             end
8421         end
8422     end
8423     first_on = nil
8424 end
8425
8426 if d == 'r' or d == 'l' then last = d end
8427 end
8428
8429 ----- IMPLICIT, REORDER -----
8430
8431 outer = save_outer
8432 last = outer
8433
8434 local state = {}
8435 state.has_r = false
8436
8437 for q = 1, #nodes do
8438     local item = nodes[q][1]
8439
8440     outer = nodes[q][3] or outer
8441
8442     local d = nodes[q][2]
8443
8444     if d == 'nsm' then d = last end          -- W1
8445     if d == 'en' then d = 'an' end
8446     local isdir = (d == 'r' or d == 'l')
8447
8448     if outer == 'l' and d == 'an' then
8449         state.san = state.san or item
8450         state.ean = item
8451     elseif state.san then
8452         head, state = insert_numeric(head, state)
8453     end
8454
8455     if outer == 'l' then
8456         if d == 'an' or d == 'r' then      -- im -> implicit
8457             if d == 'r' then state.has_r = true end
8458             state.sim = state.sim or item
8459             state.eim = item
8460         elseif d == 'l' and state.sim and state.has_r then
8461             head, state = insert_implicit(head, state, outer)
8462         elseif d == 'l' then
8463             state.sim, state.eim, state.has_r = nil, nil, false
8464         end
8465     else
8466

```

```

8467     if d == 'an' or d == 'l' then
8468         if nodes[q][3] then -- nil except after an explicit dir
8469             state.sim = item -- so we move sim 'inside' the group
8470         else
8471             state.sim = state.sim or item
8472         end
8473         state.eim = item
8474     elseif d == 'r' and state.sim then
8475         head, state = insert_implicit(head, state, outer)
8476     elseif d == 'r' then
8477         state.sim, state.eim = nil, nil
8478     end
8479 end
8480
8481 if isdir then
8482     last = d -- Don't search back - best save now
8483 elseif d == 'on' and state.san then
8484     state.san = state.san or item
8485     state.ean = item
8486 end
8487
8488 end
8489
8490 head = node.prev(head) or head
8491 % \end{macrocode}
8492 %
8493 % Now direction nodes has been distributed with relation to characters
8494 % and spaces, we need to take into account \TeX-specific elements in
8495 % the node list, to move them at an appropriate place. Firstly, with
8496 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8497 % that the latter are still discardable.
8498 %
8499 % \begin{macrocode}
8500 --- FIXES ---
8501 if has_hyperlink then
8502     local flag, linking = 0, 0
8503     for item in node.traverse(head) do
8504         if item.id == DIR then
8505             if item.dir == '+TRT' or item.dir == '+TLT' then
8506                 flag = flag + 1
8507             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8508                 flag = flag - 1
8509             end
8510             elseif item.id == 8 and item.subtype == 19 then
8511                 linking = flag
8512             elseif item.id == 8 and item.subtype == 20 then
8513                 if linking > 0 then
8514                     if item.prev.id == DIR and
8515                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8516                         d = node.new(DIR)
8517                         d.dir = item.prev.dir
8518                         node.remove(head, item.prev)
8519                         node.insert_after(head, item, d)
8520                     end
8521                 end
8522                 linking = 0
8523             end
8524         end
8525     end
8526
8527     for item in node.traverse_id(10, head) do
8528         local p = item
8529         local flag = false

```

```

8530     while p.prev and p.prev.id == 14 do
8531         flag = true
8532         p = p.prev
8533     end
8534     if flag then
8535         node.insert_before(head, p, node.copy(item))
8536         node.remove(head,item)
8537     end
8538 end
8539
8540 return head
8541 end

8542 function Babel.unset_atdir(head)
8543     local ATDIR = Babel.attr_dir
8544     for item in node.traverse(head) do
8545         node.set_attribute(item, ATDIR, 0x80)
8546     end
8547     return head
8548 end
8549 </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.

```

8550 <*\nil>
8551 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8552 \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.

```

8553 \ifx\l@nil\undefined
8554     \newlanguage\l@nil
8555     \namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8556     \let\bbl@elt\relax
8557     \edef\bbl@languages{% Add it to the list of languages
8558         \bbl@languages\bbl@elt{nil}{the\l@nil}}}}
8559 \fi

```

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

```

8560 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

\captionnil

\datenil

```
8561 \let\captionsnil\@empty
8562 \let\datenil\@empty
```

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

```
8563 \def\bbl@inidata@nil{%
8564   \bbl@elt{identification}{tag.ini}{und}%
8565   \bbl@elt{identification}{load.level}{0}%
8566   \bbl@elt{identification}{charset}{utf8}%
8567   \bbl@elt{identification}{version}{1.0}%
8568   \bbl@elt{identification}{date}{2022-05-16}%
8569   \bbl@elt{identification}{name.local}{nil}%
8570   \bbl@elt{identification}{name.english}{nil}%
8571   \bbl@elt{identification}{name.babel}{nil}%
8572   \bbl@elt{identification}{tag.bcp47}{und}%
8573   \bbl@elt{identification}{language.tag.bcp47}{und}%
8574   \bbl@elt{identification}{tag.opentype}{dflt}%
8575   \bbl@elt{identification}{script.name}{Latin}%
8576   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8577   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8578   \bbl@elt{identification}{level}{1}%
8579   \bbl@elt{identification}{encodings}{}%
8580   \bbl@elt{identification}{derivate}{no}}
8581 \@namedef{bbl@tbc@nil}{und}
8582 \@namedef{bbl@lbc@nil}{und}
8583 \@namedef{bbl@casing@nil}{und}
8584 \@namedef{bbl@lotf@nil}{dflt}
8585 \@namedef{bbl@elname@nil}{nil}
8586 \@namedef{bbl@lname@nil}{nil}
8587 \@namedef{bbl@esname@nil}{Latin}
8588 \@namedef{bbl@sname@nil}{Latin}
8589 \@namedef{bbl@sbc@nil}{Latn}
8590 \@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.

```
8591 \ldf@finish{nil}
8592 </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.

```
8593 <<Compute Julian day>> ≡
8594 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8595 \def\bbl@cs@gregleap#1{%
8596   (\bbl@fpmmod{#1}{4} == 0) &&
8597   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8598 \def\bbl@cs@jd#1#2#3{% year, month, day
8599   \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8600     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8601     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8602     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }
8603 <</Compute Julian day>>
```

13.1. Islamic

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

```
8604 <ca-islamic>
8605 <@Compute Julian day@>
```

```

8606 % == islamic (default)
8607 % Not yet implemented
8608 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar.

```

8609 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8610 ((#3 + ceil(29.5 * (#2 - 1)) +
8611 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8612 1948439.5) - 1) }
8613 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8614 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8615 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8616 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8617 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8618 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8619 \edef\bbl@tempa{%
8620 \fpeval{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8621 \edef#5{%
8622 \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8623 \edef#6{\fpeval{
8624 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8625 \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).

```

8626 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8627 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8628 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8629 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8630 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8631 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8632 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8633 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8634 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8635 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8636 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8637 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8638 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8639 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8640 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8641 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8642 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8643 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8644 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8645 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8646 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8647 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8648 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8649 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8650 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8651 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8652 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8653 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8654 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8655 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8656 65401,65431,65460,65490,65520}
8657 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8658 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8659 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8660 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8661 \ifnum#2>2014 \ifnum#2<2038
8662 \bbl@afterfi\expandafter\@gobble

```

```

8663 \fi\fi
8664 {\bbl@error{year-out-range}{2014-2038}{}}}%
8665 \edef\bbl@tempd{\fpeval{ % (Julian) day
8666 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8667 \count@\@ne
8668 \bbl@foreach\bbl@cs@umalqura@data{%
8669 \advance\count@\@ne
8670 \ifnum##1>\bbl@tempd\else
8671 \edef\bbl@tempe{\the\count@}%
8672 \edef\bbl@tempb{##1}%
8673 \fi}%
8674 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8675 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1 ) / 12) }}% annus
8676 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8677 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8678 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8679 \bbl@add\bbl@precalendar{%
8680 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8681 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8682 \bbl@replace\bbl@ld@calendar{+}{}}%
8683 \bbl@replace\bbl@ld@calendar{-}{}}%
8684 /ca-islamic

```

13.2. Hebrew

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

```

8685 \(\*ca-hebrew\)
8686 \newcount\bbl@cntcommon
8687 \def\bbl@remainder#1#2#3{%
8688 #3=#1\relax
8689 \divide #3 by #2\relax
8690 \multiply #3 by -#2\relax
8691 \advance #3 by #1\relax}%
8692 \newif\ifbbl@divisible
8693 \def\bbl@checkifdivisible#1#2{%
8694 {\countdef\tmp=0
8695 \bbl@remainder{#1}{#2}{\tmp}%
8696 \ifnum \tmp=0
8697 \global\bbl@divisibletrue
8698 \else
8699 \global\bbl@divisiblefalse
8700 \fi}}
8701 \newif\ifbbl@gregleap
8702 \def\bbl@ifgregleap#1{%
8703 \bbl@checkifdivisible{#1}{4}%
8704 \ifbbl@divisible
8705 \bbl@checkifdivisible{#1}{100}%
8706 \ifbbl@divisible
8707 \bbl@checkifdivisible{#1}{400}%
8708 \ifbbl@divisible
8709 \bbl@gregleaptrue
8710 \else
8711 \bbl@gregleapfalse
8712 \fi
8713 \else
8714 \bbl@gregleaptrue
8715 \fi
8716 \else
8717 \bbl@gregleapfalse
8718 \fi
8719 \ifbbl@gregleap}

```

```

8720 \def\bbl@gregdayspriormonths#1#2#3{%
8721     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8722         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8723     \bbl@ifggregleap{#2}%
8724     \ifnum #1 > 2
8725         \advance #3 by 1
8726     \fi
8727 \fi
8728 \global\bbl@cntcommon=#3}%
8729 #3=\bbl@cntcommon}
8730 \def\bbl@gregdaysprioryears#1#2{%
8731     {\countdef\tmpc=4
8732     \countdef\tmpb=2
8733     \tmpb=#1\relax
8734     \advance \tmpb by -1
8735     \tmpc=\tmpb
8736     \multiply \tmpc by 365
8737     #2=\tmpc
8738     \tmpc=\tmpb
8739     \divide \tmpc by 4
8740     \advance #2 by \tmpc
8741     \tmpc=\tmpb
8742     \divide \tmpc by 100
8743     \advance #2 by -\tmpc
8744     \tmpc=\tmpb
8745     \divide \tmpc by 400
8746     \advance #2 by \tmpc
8747     \global\bbl@cntcommon=#2\relax}%
8748     #2=\bbl@cntcommon}
8749 \def\bbl@absfromgreg#1#2#3#4{%
8750     {\countdef\tmpd=0
8751     #4=#1\relax
8752     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8753     \advance #4 by \tmpd
8754     \bbl@gregdaysprioryears{#3}{\tmpd}%
8755     \advance #4 by \tmpd
8756     \global\bbl@cntcommon=#4\relax}%
8757     #4=\bbl@cntcommon}
8758 \newif\ifbbl@hebrleap
8759 \def\bbl@checkleaphebryear#1{%
8760     {\countdef\tmpa=0
8761     \countdef\tmpb=1
8762     \tmpa=#1\relax
8763     \multiply \tmpa by 7
8764     \advance \tmpa by 1
8765     \bbl@remainder{\tmpa}{19}{\tmpb}%
8766     \ifnum \tmpb < 7
8767         \global\bbl@hebrleaptrue
8768     \else
8769         \global\bbl@hebrleapfalse
8770     \fi}}
8771 \def\bbl@hebrlapsedmonths#1#2{%
8772     {\countdef\tmpa=0
8773     \countdef\tmpb=1
8774     \countdef\tmpc=2
8775     \tmpa=#1\relax
8776     \advance \tmpa by -1
8777     #2=\tmpa
8778     \divide #2 by 19
8779     \multiply #2 by 235
8780     \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8781     \tmpc=\tmpb
8782     \multiply \tmpb by 12

```

```

8783 \advance #2 by \tmpb
8784 \multiply \tmpc by 7
8785 \advance \tmpc by 1
8786 \divide \tmpc by 19
8787 \advance #2 by \tmpc
8788 \global\bbl@cntcommon=#2}%
8789 #2=\bbl@cntcommon}
8790 \def\bbl@hebreleapseddays#1#2{%
8791 {\countdef\tmpa=0
8792 \countdef\tmpb=1
8793 \countdef\tmpc=2
8794 \bbl@hebreleapsedmonths{#1}{#2}%
8795 \tmpa=#2\relax
8796 \multiply \tmpa by 13753
8797 \advance \tmpa by 5604
8798 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8799 \divide \tmpa by 25920
8800 \multiply #2 by 29
8801 \advance #2 by 1
8802 \advance #2 by \tmpa
8803 \bbl@remainder{#2}{7}{\tmpa}%
8804 \ifnum \tmpc < 19440
8805 \ifnum \tmpc < 9924
8806 \else
8807 \ifnum \tmpa=2
8808 \bbl@checkleaphebyear{#1}% of a common year
8809 \ifbbl@hebrleap
8810 \else
8811 \advance #2 by 1
8812 \fi
8813 \fi
8814 \fi
8815 \ifnum \tmpc < 16789
8816 \else
8817 \ifnum \tmpa=1
8818 \advance #1 by -1
8819 \bbl@checkleaphebyear{#1}% at the end of leap year
8820 \ifbbl@hebrleap
8821 \advance #2 by 1
8822 \fi
8823 \fi
8824 \fi
8825 \else
8826 \advance #2 by 1
8827 \fi
8828 \bbl@remainder{#2}{7}{\tmpa}%
8829 \ifnum \tmpa=0
8830 \advance #2 by 1
8831 \else
8832 \ifnum \tmpa=3
8833 \advance #2 by 1
8834 \else
8835 \ifnum \tmpa=5
8836 \advance #2 by 1
8837 \fi
8838 \fi
8839 \fi
8840 \global\bbl@cntcommon=#2\relax}%
8841 #2=\bbl@cntcommon}
8842 \def\bbl@daysinhebyear#1#2{%
8843 {\countdef\tmpe=12
8844 \bbl@hebreleapseddays{#1}{\tmpe}%
8845 \advance #1 by 1

```

```

8846 \bbl@hebreleaseddays{#1}{#2}%
8847 \advance #2 by -\tmpe
8848 \global\bbl@cntcommon=#2}%
8849 #2=\bbl@cntcommon}
8850 \def\bbl@hebrdayspriormonths#1#2#3{%
8851 {\countdef\tmpf= 14
8852 #3=\ifcase #1
8853 0 \or
8854 0 \or
8855 30 \or
8856 59 \or
8857 89 \or
8858 118 \or
8859 148 \or
8860 148 \or
8861 177 \or
8862 207 \or
8863 236 \or
8864 266 \or
8865 295 \or
8866 325 \or
8867 400
8868 \fi
8869 \bbl@checkleaphebyear{#2}%
8870 \ifbbl@hebrleap
8871 \ifnum #1 > 6
8872 \advance #3 by 30
8873 \fi
8874 \fi
8875 \bbl@daysinhebyear{#2}{\tmpf}%
8876 \ifnum #1 > 3
8877 \ifnum \tmpf=353
8878 \advance #3 by -1
8879 \fi
8880 \ifnum \tmpf=383
8881 \advance #3 by -1
8882 \fi
8883 \fi
8884 \ifnum #1 > 2
8885 \ifnum \tmpf=355
8886 \advance #3 by 1
8887 \fi
8888 \ifnum \tmpf=385
8889 \advance #3 by 1
8890 \fi
8891 \fi
8892 \global\bbl@cntcommon=#3\relax}%
8893 #3=\bbl@cntcommon}
8894 \def\bbl@absfromhebr#1#2#3#4{%
8895 {#4=#1\relax
8896 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8897 \advance #4 by #1\relax
8898 \bbl@hebreleaseddays{#3}{#1}%
8899 \advance #4 by #1\relax
8900 \advance #4 by -1373429
8901 \global\bbl@cntcommon=#4\relax}%
8902 #4=\bbl@cntcommon}
8903 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8904 {\countdef\tmpx= 17
8905 \countdef\tmpy= 18
8906 \countdef\tmpz= 19
8907 #6=#3\relax
8908 \global\advance #6 by 3761

```

```

8909 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8910 \tmpz=1 \tmpy=1
8911 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8912 \ifnum \tmpx > #4\relax
8913 \global\advance #6 by -1
8914 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8915 \fi
8916 \advance #4 by -\tmpx
8917 \advance #4 by 1
8918 #5=#4\relax
8919 \divide #5 by 30
8920 \loop
8921 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8922 \ifnum \tmpx < #4\relax
8923 \advance #5 by 1
8924 \tmpy=\tmpx
8925 \repeat
8926 \global\advance #5 by -1
8927 \global\advance #4 by -\tmpy}}
8928 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8929 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8930 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8931 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8932 \bbl@hebrfromgreg
8933 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8934 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8935 \edef#4{\the\bbl@hebryear}%
8936 \edef#5{\the\bbl@hebrmonth}%
8937 \edef#6{\the\bbl@hebrday}}
8938 </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).

```

8939 <#ca-persian>
8940 <@Compute Julian day@>
8941 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8942 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8943 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8944 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8945 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8946 \bbl@afterfi\expandafter\@gobble
8947 \fi\fi
8948 {\bbl@error{year-out-range}{2013-2050}{}}}%
8949 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8950 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8951 \edef\bbl@tempc{\fpeval{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8952 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8953 \ifnum\bbl@tempc<\bbl@tempb
8954 \edef\bbl@tempa{\fpeval{\bbl@tempa-1}}% go back 1 year and redo
8955 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8956 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8957 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8958 \fi
8959 \edef#4{\fpeval{\bbl@tempa-621}}% set Jalali year
8960 \edef#6{\fpeval{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8961 \edef#5{\fpeval{% set Jalali month
8962 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8963 \edef#6{\fpeval{% set Jalali day

```

```

8964      (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))))}}
8965 </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.

```

8966 <*ca-coptic>
8967 <@Compute Julian day@>
8968 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8969   \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8970   \edef\bbl@tempc{\fpeval{\bbl@tempd - 1825029.5}}%
8971   \edef#4{\fpeval{%
8972     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8973   \edef\bbl@tempc{\fpeval{%
8974     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8975   \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
8976   \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8977 </ca-coptic>
8978 <*ca-ethiopic>
8979 <@Compute Julian day@>
8980 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8981   \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8982   \edef\bbl@tempc{\fpeval{\bbl@tempd - 1724220.5}}%
8983   \edef#4{\fpeval{%
8984     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8985   \edef\bbl@tempc{\fpeval{%
8986     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8987   \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
8988   \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8989 </ca-ethiopic>

```

13.5. Julian

Based on [ReinDersh].

```

8990 <*ca-julian>
8991 <@Compute Julian day@>
8992 \def\bbl@ca@julian#1-#2-#3\@#4#5#6{%
8993   \edef\bbl@tempj{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + .5}}%
8994   \edef\bbl@tempa{\fpeval{\bbl@tempj + 32082.5}}%
8995   \edef\bbl@tempb{\fpeval{floor((4 * \bbl@tempa + 3) / 1461)}}%
8996   \edef\bbl@tempc{\fpeval{\bbl@tempa - floor(1461*\bbl@tempb/4)}}%
8997   \edef\bbl@tempd{\fpeval{floor((5 * \bbl@tempc + 2) / 153)}}%
8998   \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
8999   \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
9000   \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}%
9001 </ca-julian>

```

13.6. Buddhist

That's very simple.

```

9002 <*ca-buddhist>
9003 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
9004   \edef#4{\number\numexpr#1+543\relax}%
9005   \edef#5{#2}%
9006   \edef#6{#3}%
9007 </ca-buddhist>
9008 %
9009 % \subsection{Chinese}
9010 %
9011 % Brute force, with the Julian day of first day of each month. The
9012 % table has been computed with the help of \textsf{python-lunardate} by

```

```

9013% Ricky Yeung, GPLv2 (but the code itself has not been used). The range
9014% is 2015-2044.
9015%
9016% \begin{macrocode}
9017(*ca-chinese)
9018\ExplSyntaxOn
9019<@Compute Julian day@>
9020\def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
9021\edef\bbl@tempd{\fpeval{%
9022\bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
9023\count@z@
9024\@tempcnta=2015
9025\bbl@foreach\bbl@cs@chinese@data{%
9026\ifnum##1>\bbl@tempd\else
9027\advance\count@\@ne
9028\ifnum\count@>12
9029\count@\@ne
9030\advance\@tempcnta\@ne\fi
9031\bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
9032\ifin@
9033\advance\count@\m@ne
9034\edef\bbl@tempe{\the\numexpr\count@+12\relax}%
9035\else
9036\edef\bbl@tempe{\the\count@}%
9037\fi
9038\edef\bbl@tempb{##1}%
9039\fi}%
9040\edef#4{\the\@tempcnta}%
9041\edef#5{\bbl@tempe}%
9042\edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
9043\def\bbl@cs@chinese@leap{%
9044885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
9045\def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
9046354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
9047768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
90481152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
90491506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
90501860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
90512214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
90522569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
90532923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
90543278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
90553632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
90563987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
90574341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
90584695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
90595049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
90605403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
90615758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
90626113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
90636467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
90646821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
90657175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
90667529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
90677884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
90688239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
90698593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
90708948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
90719302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
90729656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
907310010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
907410306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
907510602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%

```

```

9076 10896,10926,10956,10986,11015,11045,11074,11103}
9077 \ExplSyntaxOff
9078 </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`.

```

9079 <{*bplain | blplain}
9080 \catcode`\{=1 % left brace is begin-group character
9081 \catcode`\}=2 % right brace is end-group character
9082 \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).

```

9083 \openin 0 hyphen.cfg
9084 \ifeof0
9085 \else
9086   \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.

```

9087 \def\input #1 {%
9088   \let\input\input
9089   \a hyphen.cfg
9090   \let\input\input
9091 }
9092 \fi
9093 </{*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`.

```

9094 <bplain>\a plain.tex
9095 <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.

```

9096 <bplain>\def\fmtname{babel-plain}
9097 <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).

```
9098 <<{*Emulate LaTeX}>> ≡
9099 \def\@empty{}
9100 \def\loadlocalcfg#1{%
9101   \openin0#1.cfg
9102   \ifeof0
9103     \closein0
9104   \else
9105     \closein0
9106     {\immediate\write16{*****}%
9107       \immediate\write16{* Local config file #1.cfg used}%
9108       \immediate\write16{*}%
9109     }
9110     \input #1.cfg\relax
9111   \fi
9112 \endofldef}
```

14.3. General tools

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

```
9113 \long\def\@firstofone#1{#1}
9114 \long\def\@firstoftwo#1#2{#1}
9115 \long\def\@secondoftwo#1#2{#2}
9116 \def\@nnil{\@nil}
9117 \def\@gobbletwo#1#2{}
9118 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
9119 \def\@staror@long#1{%
9120   \@ifstar
9121   {\let\l@ngrel@x\relax#1}%
9122   {\let\l@ngrel@x\long#1}}
9123 \let\l@ngrel@x\relax
9124 \def\@car#1#2\@nil{#1}
9125 \def\@cdr#1#2\@nil{#2}
9126 \let\@typeset@protect\relax
9127 \let\protected@edef\edef
9128 \long\def\@gobble#1{}
9129 \edef\@backslashchar{\expandafter\@gobble\string\}
9130 \def\strip@prefix#1>{}
9131 \def\g@addto@macro#1#2{{%
9132   \toks@\expandafter{#1#2}%
9133   \xdef#1{\the\toks@}}}
9134 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
9135 \def\@nameuse#1{\csname #1\endcsname}
9136 \def\@ifundefined#1{%
9137   \expandafter\ifx\csname#1\endcsname\relax
9138     \expandafter\@firstoftwo
9139   \else
9140     \expandafter\@secondoftwo
9141   \fi}
9142 \def\@expandtwoargs#1#2#3{%
9143   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9144 \def\zap@space#1 #2{%
9145   #1%
9146   \ifx#2\@empty\else\expandafter\zap@space\fi
9147   #2}
9148 \let\bbl@trace\@gobble
9149 \def\bbl@error#1{% Implicit #2#3#4}
```

```

9150 \begingroup
9151 \catcode\`=0 \catcode\==12 \catcode\`=12
9152 \catcode\^^M=5 \catcode\%=14
9153 \input errbabel.def
9154 \endgroup
9155 \bbl@error{#1}}
9156 \def\bbl@warning#1{%
9157 \begingroup
9158 \newlinechar=\^^J
9159 \def\{\^^J(babel) }%
9160 \message{\`#1}%
9161 \endgroup}
9162 \let\bbl@infowarn\bbl@warning
9163 \def\bbl@info#1{%
9164 \begingroup
9165 \newlinechar=\^^J
9166 \def\{\^^J}%
9167 \wlog{#1}%
9168 \endgroup}

```

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

```

9169 \ifx\@preamblecmds\undefined
9170 \def\@preamblecmds{}
9171 \fi
9172 \def\@onlypreamble#1{%
9173 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9174 \@preamblecmds\do#1}}
9175 \@onlypreamble\@onlypreamble

```

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

```

9176 \def\begindocument{%
9177 \@begindocumenthook
9178 \global\let\@begindocumenthook\undefined
9179 \def\do##1{\global\let##1\undefined}%
9180 \@preamblecmds
9181 \global\let\do\noexpand}
9182 \ifx\@begindocumenthook\undefined
9183 \def\@begindocumenthook{}
9184 \fi
9185 \@onlypreamble\@begindocumenthook
9186 \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`.

```

9187 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
9188 \@onlypreamble\AtEndOfPackage
9189 \def\@endoflfd{}
9190 \@onlypreamble\@endoflfd
9191 \let\bbl@afterlang\@empty
9192 \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.

```

9193 \catcode\&=\z@
9194 \ifx\if@files\undefined
9195 \expandafter\let\csname if@files\expandafter\endcsname
9196 \csname iffalse\endcsname
9197 \fi
9198 \catcode\&=4

```

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

```

9199 \def\newcommand{\@star@or@long\new@command}
9200 \def\new@command#1{%
9201   \@testopt{\@newcommand#1}0}
9202 \def\@newcommand#1[#2]{%
9203   \@ifnextchar [{\@xargdef#1[#2]}%
9204     {\@argdef#1[#2]}}
9205 \long\def\@argdef#1[#2]#3{%
9206   \@yargdef#1\@ne{#2}{#3}}
9207 \long\def\@xargdef#1[#2][#3]#4{%
9208   \expandafter\def\expandafter#1\expandafter{%
9209     \expandafter\@protected@testopt\expandafter #1%
9210     \csname\string#1\expandafter\endcsname{#3}}}%
9211   \expandafter\@yargdef \csname\string#1\endcsname
9212   \tw@{#2}{#4}}
9213 \long\def\@yargdef#1#2#3{%
9214   \@tempcnta#3\relax
9215   \advance \@tempcnta \@ne
9216   \let\@hash@\relax
9217   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9218   \@tempcntb #2%
9219   \@whilenum\@tempcntb <\@tempcnta
9220   \do{%
9221     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
9222     \advance\@tempcntb \@ne}%
9223   \let\@hash@###%
9224   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9225 \def\providecommand{\@star@or@long\provide@command}
9226 \def\provide@command#1{%
9227   \begingroup
9228     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9229   \endgroup
9230   \expandafter\ifundefined\@gtempa
9231     {\def\reserved@a{\new@command#1}}%
9232     {\let\reserved@a\relax
9233       \def\reserved@a{\new@command\reserved@a}}%
9234   \reserved@a}%

9235 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9236 \def\declare@robustcommand#1{%
9237   \edef\reserved@a{\string#1}%
9238   \def\reserved@b{#1}%
9239   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9240   \edef#1{%
9241     \ifx\reserved@a\reserved@b
9242       \noexpand\x@protect
9243       \noexpand#1%
9244     \fi
9245     \noexpand\protect
9246     \expandafter\noexpand\csname
9247       \expandafter\@gobble\string#1 \endcsname
9248   }%
9249   \expandafter\new@command\csname
9250     \expandafter\@gobble\string#1 \endcsname
9251 }
9252 \def\x@protect#1{%
9253   \ifx\protect\@typeset@protect\else
9254     \x@protect#1%
9255   \fi
9256 }
9257 \catcode`\&=\z@ % Trick to hide conditionals
9258 \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`.

```

9259 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9260 \catcode`\&=4
9261 \ifx\in@\@undefined
9262 \def\in@#1#2{%
9263 \def\in@@##1#1##2##3\in@@{%
9264 \ifx\in@@##2\in@false\else\in@true\fi}%
9265 \in@@##2#1\in@\in@@}
9266 \else
9267 \let\bbl@tempa\@empty
9268 \fi
9269 \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).

```

9270 \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.

```

9271 \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.

```

9272 \ifx\@tempcnta\@undefined
9273 \csname newcount\endcsname\@tempcnta\relax
9274 \fi
9275 \ifx\@tempcntb\@undefined
9276 \csname newcount\endcsname\@tempcntb\relax
9277 \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`).

```

9278 \ifx\bye\@undefined
9279 \advance\count10 by -2\relax
9280 \fi
9281 \ifx\@ifnextchar\@undefined
9282 \def\@ifnextchar#1#2#3{%
9283 \let\reserved@d=#1%
9284 \def\reserved@a{#2}\def\reserved@b{#3}%
9285 \futurelet\@let@token\@ifnch}
9286 \def\@ifnch{%
9287 \ifx\@let@token\@sptoken
9288 \let\reserved@c\@xifnch
9289 \else
9290 \ifx\@let@token\reserved@d
9291 \let\reserved@c\reserved@a
9292 \else
9293 \let\reserved@c\reserved@b
9294 \fi
9295 \fi
9296 \reserved@c}
9297 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
9298 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
9299 \fi
9300 \def\@testopt#1#2{%
9301 \@ifnextchar[#{1}{#1[#{2}]}}
9302 \def\@protected@testopt#1{%
9303 \ifx\protect\@typeset@protect
9304 \expandafter\@testopt

```

```

9305 \else
9306 \@@protect#1%
9307 \fi}
9308 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9309 #2\relax}\fi}
9310 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9311 \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9312 \def\DeclareTextCommand{%
9313 \@@dec@text@cmd\providecommand
9314 }
9315 \def\ProvideTextCommand{%
9316 \@@dec@text@cmd\providecommand
9317 }
9318 \def\DeclareTextSymbol#1#2#3{%
9319 \@@dec@text@cmd\chardef#1{#2}#3\relax
9320 }
9321 \def\@dec@text@cmd#1#2#3{%
9322 \expandafter\def\expandafter#2%
9323 \expandafter{%
9324 \csname#3-cmd\expandafter\endcsname
9325 \expandafter#2%
9326 \csname#3\string#2\endcsname
9327 }%
9328 % \let\@ifdefinable\@rc@ifdefinable
9329 \expandafter#1\csname#3\string#2\endcsname
9330 }
9331 \def\@current@cmd#1{%
9332 \ifx\protect\@typeset@protect\else
9333 \noexpand#1\expandafter\@gobble
9334 \fi
9335 }
9336 \def\@changed@cmd#1#2{%
9337 \ifx\protect\@typeset@protect
9338 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9339 \expandafter\ifx\csname ?\string#1\endcsname\relax
9340 \expandafter\def\csname ?\string#1\endcsname{%
9341 \@changed@x@err{#1}%
9342 }%
9343 \fi
9344 \global\expandafter\let
9345 \csname\cf@encoding\string#1\expandafter\endcsname
9346 \csname ?\string#1\endcsname
9347 \fi
9348 \csname\cf@encoding\string#1%
9349 \expandafter\endcsname
9350 \else
9351 \noexpand#1%
9352 \fi
9353 }
9354 \def\@changed@x@err#1{%
9355 \errhelp{Your command will be ignored, type <return> to proceed}%
9356 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9357 \def\DeclareTextCommandDefault#1{%
9358 \DeclareTextCommand#1?%
9359 }
9360 \def\ProvideTextCommandDefault#1{%
9361 \ProvideTextCommand#1?%
9362 }
9363 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd

```

```

9364 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9365 \def\DeclareTextAccent#1#2#3{%
9366   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9367 }
9368 \def\DeclareTextCompositeCommand#1#2#3#4{%
9369   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9370   \edef\reserved@b{\string##1}%
9371   \edef\reserved@c{%
9372     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9373   \ifx\reserved@b\reserved@c
9374     \expandafter\expandafter\expandafter\ifx
9375       \expandafter\@car\reserved@a\relax\relax\@nil
9376       \@text@composite
9377   \else
9378     \edef\reserved@b##1{%
9379       \def\expandafter\noexpand
9380         \csname#2\string#1\endcsname###1{%
9381           \noexpand\@text@composite
9382             \expandafter\noexpand\csname#2\string#1\endcsname
9383             ###1\noexpand\@empty\noexpand\@text@composite
9384             {##1}%
9385       }%
9386     }%
9387     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9388   \fi
9389   \expandafter\def\csname\expandafter\string\csname
9390     #2\endcsname\string#1-\string#3\endcsname{#4}
9391 \else
9392   \errhelp{Your command will be ignored, type <return> to proceed}%
9393   \errmessage{\string\DeclareTextCompositeCommand\space used on
9394     inappropriate command \protect#1}
9395 \fi
9396 }
9397 \def\@text@composite#1#2#3\@text@composite{%
9398   \expandafter\@text@composite@x
9399   \csname\string#1-\string#2\endcsname
9400 }
9401 \def\@text@composite@x#1#2{%
9402   \ifx#1\relax
9403     #2%
9404   \else
9405     #1%
9406   \fi
9407 }
9408 %
9409 \def\@strip@args#1:#2-#3\@strip@args{#2}
9410 \def\DeclareTextComposite#1#2#3#4{%
9411   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9412   \bgroup
9413     \lccode`\@=#4%
9414     \lowercase{%
9415       \egroup
9416       \reserved@a @%
9417     }%
9418 }
9419 %
9420 \def\UseTextSymbol#1#2{#2}
9421 \def\UseTextAccent#1#2#3{}
9422 \def\@use@text@encoding#1{}
9423 \def\DeclareTextSymbolDefault#1#2{%
9424   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}{#1}}%
9425 }
9426 \def\DeclareTextAccentDefault#1#2{%

```

```

9427 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9428 }
9429 \def\cf@encoding{OT1}

```

Currently we only use the $\TeX$ 2 ϵ method for accents for those that are known to be made active in *some* language definition file.

```

9430 \DeclareTextAccent{"}{OT1}{127}
9431 \DeclareTextAccent{'}{OT1}{19}
9432 \DeclareTextAccent{^}{OT1}{94}
9433 \DeclareTextAccent{\`}{OT1}{18}
9434 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9435 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9436 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
9437 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9438 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
9439 \DeclareTextSymbol{\i}{OT1}{16}
9440 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9441 \ifx\scriptsize\undefined
9442 \let\scriptsize\sevenrm
9443 \fi

```

And a few more “dummy” definitions.

```

9444 \def\language{english}%
9445 \let\bbl@opt@shorthands\@nnil
9446 \def\bbl@ifshorthand#1#2#3{#2}%
9447 \let\bbl@language@opts\@empty
9448 \let\bbl@provide@locale\relax
9449 \ifx\babeloptionstrings\undefined
9450 \let\bbl@opt@strings\@nnil
9451 \else
9452 \let\bbl@opt@strings\babeloptionstrings
9453 \fi
9454 \def\BabelStringsDefault{generic}
9455 \def\bbl@tempa{normal}
9456 \ifx\babeloptionmath\bbl@tempa
9457 \def\bbl@mathnormal{\noexpand\textormath}
9458 \fi
9459 \def\AfterBabelLanguage#1#2{}
9460 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9461 \let\bbl@afterlang\relax
9462 \def\bbl@opt@safe{BR}
9463 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
9464 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9465 \expandafter\newif\csname ifbbl@single\endcsname
9466 \chardef\bbl@bidimode\z@
9467 <</Emulate LaTeX>>

```

A proxy file:

```

9468 <*\plain>
9469 \input babel.def
9470 </\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 L^AT_EX styles*, *TUGboat* 10 (1989) #3, pp. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The T_EXbook*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *L^AT_EX, A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: T_EXhax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German T_EX*, *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).