

Babel

Code

Version 25.17
2025/12/11

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	25
4.5	Compatibility with language.def	25
4.6	Hooks	26
4.7	Setting up language files	27
4.8	Shorthands	29
4.9	Language attributes	38
4.10	Support for saving and redefining macros	40
4.11	French spacing	41
4.12	Hyphens	41
4.13	Multiencoding strings	43
4.14	Tailor captions	48
4.15	Making glyphs available	49
4.15.1	Quotation marks	49
4.15.2	Letters	51
4.15.3	Shorthands for quotation marks	51
4.15.4	Umlauts and tremas	52
4.16	Layout	54
4.17	Load engine specific macros	54
4.18	Creating and modifying languages	54
4.19	Main loop in ‘provide’	62
4.20	Processing keys in ini	66
4.21	French spacing (again)	72
4.22	Handle language system	73
4.23	Numerals	74
4.24	Casing	75
4.25	Getting info	76
4.26	BCP 47 related commands	77
5	Adjusting the Babel behavior	78
5.1	Cross referencing macros	80
5.2	Layout	83
5.3	Marks	84
5.4	Other packages	85
5.4.1	ifthen	85
5.4.2	varioref	86
5.4.3	hhline	86
5.5	Encoding and fonts	87
5.6	Basic bidi support	89
5.7	Local Language Configuration	92
5.8	Language options	92

6	The kernel of Babel	96
7	Error messages	96
8	Loading hyphenation patterns	100
9	luatex + xetex: common stuff	104
10	Hooks for XeTeX and LuaTeX	107
10.1	XeTeX	107
10.2	Support for interchar	109
10.3	Layout	111
10.4	8-bit TeX	113
10.5	LuaTeX	113
10.6	Southeast Asian scripts	120
10.7	CJK line breaking	121
10.8	Arabic justification	123
10.9	Common stuff	128
10.10	Automatic fonts and ids switching	128
10.11	Bidi	135
10.12	Layout	137
10.13	Lua: transforms	147
10.14	Lua: Auto bidi with basic and basic-r	157
11	Data for CJK	168
12	The ‘nil’ language	169
13	Calendars	170
13.1	Islamic	170
13.2	Hebrew	171
13.3	Persian	175
13.4	Coptic and Ethiopic	176
13.5	Julian	176
13.6	Buddhist	177
14	Support for Plain T_EX (plain.def)	178
14.1	Not renaming hyphen.tex	178
14.2	Emulating some L ^A T _E X features	179
14.3	General tools	179
14.4	Encoding related macros	183
15	Acknowledgements	186

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=25.17>>
2 <<date=2025/12/11>>
```

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{macrode}
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 % Don't use. Experimental.
367 \newif\ifbbl@single
368 \DeclareOption{selectors=off}{\bbl@singletrue}
369 <@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.

```

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

```

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

```

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

```

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

```

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

```

390 \ProcessOptions*

```

3.5. Post-process some options

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

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

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

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

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

```
422 \edef\bbl@opt@shorthands{%
423 \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.

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

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

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

```

436 \def\bbl@opt@safe{BR}
437 % \let\bbl@opt@safe\@empty % Pending of \cite
438 \fi

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

```

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

```

468 < *package | core >
469 \def\bbl@version{<@version@>}
470 \def\bbl@date{<@date@>}
471 <@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.

```

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

```

```

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

```

487 \def\bbl@fixname#1{%
488   \begingroup
489   \def\bbl@tempe{l@}%
490   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
491   \bbl@tempd
492     {\lowercase\expandafter{\bbl@tempd}%
493     {\uppercase\expandafter{\bbl@tempd}%
494     \@empty
495     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
496     {\uppercase\expandafter{\bbl@tempd}}}%
497     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
498     {\lowercase\expandafter{\bbl@tempd}}}%
499     \@empty
500   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
501   \bbl@tempd
502   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
503 \def\bbl@iflanguage#1{%
504   \@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, with \bbl@bcpcase, casing is the correct one, so that sr-latn-ba becomes fr-Latn-BA. Note #4 may contain some \@empty’s, but they are eventually removed.

\bbl@bcpllookup either returns the found ini tag or it is \relax.

```

505 \def\bbl@bcpcase#1#2#3#4\@#5{%
506   \ifx\@empty#3%
507     \uppercase{\def#5{#1#2}}%
508   \else
509     \uppercase{\def#5{#1}}%
510     \lowercase{\edef#5{#5#2#3#4}}%
511   \fi}
512 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
513   \let\bbl@bcp\relax
514   \lowercase{\def\bbl@tempa{#1}}%
515   \ifx\@empty#2%
516     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
517   \else\ifx\@empty#3%
518     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb}
519     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
520     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
521     {}%
522   \ifx\bbl@bcp\relax
523     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
524   \fi
525   \else
526     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb}
527     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc}
528     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
529     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
530     {}%

```

```

531 \ifx\bb@bcp\relax
532 \IfFileExists{babel-\bb@tempa-\bb@tempc.ini}%
533 {\edef\bb@bcp{\bb@tempa-\bb@tempc}}}%
534 {}%
535 \fi
536 \ifx\bb@bcp\relax
537 \IfFileExists{babel-\bb@tempa-\bb@tempc.ini}%
538 {\edef\bb@bcp{\bb@tempa-\bb@tempc}}}%
539 {}%
540 \fi
541 \ifx\bb@bcp\relax
542 \IfFileExists{babel-\bb@tempa.ini}{\let\bb@bcp\bb@tempa}}}%
543 \fi
544 \fi\fi}
545 \let\bb@initoload\relax

```

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

```

546 \def\iflanguage#1{%
547 \bb@iflanguage{#1}{%
548 \ifnum\csname l@#1\endcsname=\language
549 \expandafter\@firstoftwo
550 \else
551 \expandafter\@secondoftwo
552 \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.

```

553 \let\bb@select@type\z@
554 \edef\selectlanguage{%
555 \noexpand\protect
556 \expandafter\noexpand\csname selectlanguage \endcsname}

```

Because the command `\selectlanguage` could be used in a moving argument it expands to `\protect\selectlanguage`. Therefore, we have to make sure that a macro `\protect` exists. If it doesn't it is `\let` to `\relax`.

```

557 \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.

```

558 \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.

\bb@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 `\bb@pop@language` to be executed at the end of the group. It calls `\bb@set@language` with the name of the current language as its argument.

\bb@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 `\bb@language@stack` and initially empty.

```

559 \def\bb@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:

```
560 \def\bbl@push@language{%
561   \ifx\language\undefined\else
562     \ifx\currentgrouplevel\undefined
563       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
564     \else
565       \ifnum\currentgrouplevel=\z@
566         \xdef\bbl@language@stack{\language+}%
567       \else
568         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
569       \fi
570     \fi
571 }
```

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.

```
572 \def\bbl@pop@lang#1+#2\@@{%
573   \edef\language{#1}%
574   \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).

```
575 \let\bbl@ifrestoring\@secondoftwo
576 \def\bbl@pop@language{%
577   \expandafter\bbl@pop@lang\bbl@language@stack\@@
578   \let\bbl@ifrestoring\@firstoftwo
579   \expandafter\bbl@set@language\expandafter{\language}%
580   \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).

```
581 \chardef\localeid\z@
582 \gdef\bbl@id@last{0} % No real need for a new counter
583 \def\bbl@id@assign{%
584   \bbl@ifunset\bbl@id@\language}%
585   {\count@\bbl@id@last\relax
586    \advance\count@\@ne
587    \global\bbl@csarg\chardef{id@\language}\count@
588    \xdef\bbl@id@last{\the\count@}%
589    \ifcase\bbl@engine\or
590      \directlua{
591        Babel.locale_props[\bbl@id@last] = {}
592        Babel.locale_props[\bbl@id@last].name = '\language'
593        Babel.locale_props[\bbl@id@last].vars = {}
594      }%
595    \fi}%
596  }%
597  \chardef\localeid\bbl@c{l{id@}}
```

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

```

598 \let\bbl@select@opts\@empty
599 \expandafter\def\csname selectlanguage \endcsname{%
600   \ifnextchar[\bbl@select@s{\bbl@select@s[]}}
601 \def\bbl@select@s[#1]#2{%
602   \def\bbl@select@opts{#1}%
603   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@ \fi
604   \bbl@push@language
605   \aftergroup\bbl@pop@language
606   \bbl@set@language{#2}}
607 \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).

```

608 \def\BabelContentsFiles{toc,lof,lot}
609 \def\bbl@set@language#1{% from selectlanguage, pop@
610   % The old buggy way. Preserved for compatibility, but simplified
611   \edef\language{\expandafter\string#1\@empty}%
612   \select@language{\language}%
613   \bbl@xin@{,main,}{,\bbl@select@opts,}%
614   \ifin@
615     \let\bbl@main@language\localename
616     \let\mainlocalename\localename
617   \fi
618   \let\bbl@select@opts\@empty
619   % write to auxs
620   \expandafter\ifx\csname date\language\endcsname\relax\else
621     \if@filesw
622       \bbl@xin@{,noaux,}{,\bbl@select@opts,}%
623       \ifin@\else
624         \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
625           \bbl@savelastskip
626           \protected@write\auxout{}\string\babel@aux{\bbl@auxname}{}}%
627           \bbl@restorelastskip
628         \fi
629       \fi
630       \bbl@usehooks{write}{}%
631     \fi
632   \fi}
633 %
634 \let\bbl@restorelastskip\relax
635 \let\bbl@savelastskip\relax
636 %
637 \def\select@language#1{% from set@, babel@aux, babel@toc
638   \ifx\bbl@select@name\@empty
639     \def\bbl@select@name{select}%
640   \fi
641   % set hymap
642   \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
643   % set name (when coming from babel@aux)
644   \edef\language{#1}%
645   \bbl@fixname\language
646   % define \localename when coming from set@, with a trick
647   \ifx\scantokens\@undefined

```

```

648 \def\localename{??}%
649 \else
650 \bbl@exp{\scantokens{\def\localename{\language}\noexpand}\relax}%
651 \fi
652 \bbl@provide@locale
653 \bbl@iflanguage\language{\%
654 \let\bbl@select@type\z@
655 \expandafter\bbl@switch\expandafter{\language}}
656 \def\babel@aux#1#2{%
657 \select@language{#1}%
658 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
659 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}%
660 \def\babel@toc#1#2{%
661 \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`.

```

662 \newif\ifbbl@usedategroup
663 \let\bbl@savextras\@empty
664 \def\bbl@switch#1{% from select@, foreign@
665 % restore
666 \originalTeX
667 \expandafter\def\expandafter\originalTeX\expandafter{%
668 \csname noextras#1\endcsname
669 \let\originalTeX\@empty
670 \babel@beginsave}%
671 \bbl@usehooks{afterreset}}}%
672 \languageshorthands{none}%
673 % set the locale id
674 \bbl@id@assign
675 % switch captions, date
676 \bbl@bsphack
677 \ifcase\bbl@select@type
678 \csname captions#1\endcsname\relax
679 \csname date#1\endcsname\relax
680 \else
681 \bbl@xin@{,captions,},{,\bbl@select@opts,}%
682 \ifin@
683 \csname captions#1\endcsname\relax
684 \fi
685 \bbl@xin@{,date,},{,\bbl@select@opts,}%
686 \ifin@ % if \foreign... within \<language>date
687 \csname date#1\endcsname\relax
688 \fi
689 \fi
690 \bbl@esphack
691 % switch extras
692 \csname bbl@preextras@#1\endcsname
693 \bbl@usehooks{beforeextras}}}%
694 \csname extras#1\endcsname\relax
695 \bbl@usehooks{afterextras}}}%

```

```

696 % > babel-ensure
697 % > babel-sh-<short>
698 % > babel-bidi
699 % > babel-fontspec
700 \let\bbbl@savedextras\@empty
701 % hyphenation - case mapping
702 \ifcase\bbbl@opt@hyphenmap\or
703   \def\BabelLower##1##2{\lccode##1=##2\relax}%
704   \ifnum\bbbl@hymapsel>4\else
705     \csname\language\ @bbbl@hyphenmap\endcsname
706   \fi
707   \chardef\bbbl@opt@hyphenmap\z@
708 \else
709   \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
710     \csname\language\ @bbbl@hyphenmap\endcsname
711   \fi
712 \fi
713 \let\bbbl@hymapsel\@cclv
714 % hyphenation - select rules
715 \ifnum\csname l@\language\endcsname=\l@unhyphenated
716   \edef\bbbl@tempa{u}%
717 \else
718   \edef\bbbl@tempa{\bbbl@cl{\lnbrk}}%
719 \fi
720 % linebreaking - handle u, e, k (v in the future)
721 \bbbl@xin@{/u}{/\bbbl@tempa}%
722 \ifin@ \else \bbbl@xin@{/e}{/\bbbl@tempa} \fi % elongated forms
723 \ifin@ \else \bbbl@xin@{/k}{/\bbbl@tempa} \fi % only kashida
724 \ifin@ \else \bbbl@xin@{/p}{/\bbbl@tempa} \fi % padding (e.g., Tibetan)
725 \ifin@ \else \bbbl@xin@{/v}{/\bbbl@tempa} \fi % variable font
726 % hyphenation - save mins
727 \babel@savevariable\lefthyphenmin
728 \babel@savevariable\righthyphenmin
729 \ifnum\bbbl@engine=\@ne
730   \babel@savevariable\hyphenationmin
731 \fi
732 \ifin@
733   % unhyphenated/kashida/elongated/padding = allow stretching
734   \language\l@unhyphenated
735   \babel@savevariable\emergencystretch
736   \emergencystretch\maxdimen
737   \babel@savevariable\hbadness
738   \hbadness\@M
739 \else
740   % other = select patterns
741   \bbbl@patterns{#1}%
742 \fi
743 % hyphenation - set mins
744 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
745   \set@hyphenmins\tw@\thr@@\relax
746   \@nameuse{bbbl@hyphenmins@}%
747 \else
748   \expandafter\expandafter\expandafter\set@hyphenmins
749     \csname #1hyphenmins\endcsname\relax
750 \fi
751 \@nameuse{bbbl@hyphenmins@}%
752 \@nameuse{bbbl@hyphenmins@\language}%
753 \@nameuse{bbbl@hyphenatmin@}%
754 \@nameuse{bbbl@hyphenatmin@\language}%
755 \let\bbbl@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.

```
756 \long\def\otherlanguage#1{%
757   \def\bbl@selectorname{other}%
758   \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\thr@@\fi
759   \csname selectlanguage \endcsname{#1}%
760   \ignorespaces}
```

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

```
761 \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`.

```
762 \expandafter\def\csname otherlanguage*\endcsname{%
763   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
764 \def\bbl@otherlanguage@s[#1]#2{%
765   \def\bbl@selectorname{other*}%
766   \ifnum\bbl@hymapsel=\ccclv\chardef\bbl@hymapsel4\relax\fi
767   \def\bbl@select@opts{#1}%
768   \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”.

```
769 \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.

```
770 \providecommand\bbl@beforeforeign{}
771 \edef\foreignlanguage{%
772   \noexpand\protect
773   \expandafter\noexpand\csname foreignlanguage \endcsname}
774 \expandafter\def\csname foreignlanguage \endcsname{%
775   \@ifstar\bbl@foreign@s\bbl@foreign@x}
776 \providecommand\bbl@foreign@x[3][]{%
777   \begingroup
778     \def\bbl@selectorname{foreign}%
779     \def\bbl@select@opts{#1}%
780     \let\BabelText\@firstofone
781     \bbl@beforeforeign
782     \foreign@language{#2}%
783     \bbl@usehooks{foreign}{}%
784     \BabelText{#3}% Now in horizontal mode!
785   \endgroup}
```

```

786 \def\bbl@foreign@s#1#2{%
787   \begingroup
788     {\par}%
789     \def\bbl@select@name{foreign*}%
790     \let\bbl@select@opts\@empty
791     \let\BabelText\@firstofone
792     \foreign@language{#1}%
793     \bbl@usehooks{foreign*}{}%
794     \bbl@dirparastext
795     \BabelText{#2}% Still in vertical mode!
796     {\par}%
797   \endgroup}
798 \providecommand\BabelWrapText[1]{%
799   \def\bbl@tempa{\def\BabelText###1}%
800   \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.

```

801 \def\foreign@language#1{%
802   % set name
803   \edef\language#1%
804   \ifbbl@usedategroup
805     \bbl@add\bbl@select@opts{,date,}%
806     \bbl@usedategroupfalse
807   \fi
808   \bbl@fixname\language
809   \let\localname\language
810   \bbl@provide@locale
811   \bbl@iflanguage\language{%
812     \let\bbl@select@type\@ne
813     \expandafter\bbl@switch\expandafter{\language}}

```

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

```

814 \def\IfBabelSelectorTF#1{%
815   \bbl@xin@{\bbl@select@name,}{,\zap@space#1 \@empty,}%
816   \ifin@
817     \expandafter\@firstoftwo
818   \else
819     \expandafter\@secondoftwo
820   \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.

```

821 \let\bbl@hyphlist\@empty
822 \let\bbl@hyphenation@\relax
823 \let\bbl@pttnlist\@empty
824 \let\bbl@patterns@\relax
825 \let\bbl@hymapsel=\ccclv
826 \def\bbl@patterns#1{%
827   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
828     \csname l@#1\endcsname
829     \edef\bbl@tempa{#1}%
830   \else
831     \csname l@#1:\f@encoding\endcsname
832     \edef\bbl@tempa{#1:\f@encoding}%

```

```

833 \fi
834 \@expandtwoargs\bbbl@usehooks{patterns}{\#1}{\bbbl@tempa}}%
835 % > luatex
836 \@ifundefined{bbbl@hyphenation@}{\% Can be \relax!
837 \begingroup
838 \bbbl@xin@{, \number\language,}{, \bbbl@hyphlist}%
839 \ifin@else
840 \@expandtwoargs\bbbl@usehooks{hyphenation}{\#1}{\bbbl@tempa}}%
841 \hyphenation{%
842 \bbbl@hyphenation@
843 \@ifundefined{bbbl@hyphenation@#1}%
844 \@empty
845 {\space\csname bbbl@hyphenation@#1\endcsname}}%
846 \xdef\bbbl@hyphlist{\bbbl@hyphlist\number\language,}%
847 \fi
848 \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 other `\language*`.

```

849 \def\hyphenrules#1{%
850 \edef\bbbl@tempf{\#1}%
851 \bbbl@fixname\bbbl@tempf
852 \bbbl@iflanguage\bbbl@tempf{%
853 \expandafter\bbbl@patterns\expandafter{\bbbl@tempf}%
854 \ifx\languageshorthands\@undefined\else
855 \languageshorthands{none}%
856 \fi
857 \expandafter\ifx\csname\bbbl@tempf hyphenmins\endcsname\relax
858 \set@hyphenmins\tw@\thr@@\relax
859 \else
860 \expandafter\expandafter\expandafter\set@hyphenmins
861 \csname\bbbl@tempf hyphenmins\endcsname\relax
862 \fi}}
863 \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.

```

864 \def\providehyphenmins#1#2{%
865 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
866 \@namedef{\#1hyphenmins}{\#2}%
867 \fi}

```

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

```

868 \def\set@hyphenmins#1#2{%
869 \lefthyphenmin#1\relax
870 \righthyphenmin#2\relax}

```

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

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

```

871 \ifx\ProvidesFile\@undefined
872 \def\ProvidesLanguage#1[#2 #3 #4]{%
873 \wlog{Language: #1 #4 #3 <#2>}%
874 }
875 \else
876 \def\ProvidesLanguage#1{%

```

```

877 \begingroup
878 \catcode\ 10 %
879 \@makeother\/%
880 \@ifnextchar[%]
881 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
882 \def\@provideslanguage#1[#2]{%
883 \wlog{Language: #1 #2}%
884 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
885 \endgroup}
886 \fi

```

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

```
887 \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.

```
888 \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’:

```

889 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
890 \let\uselocale\setlocale
891 \let\locale\setlocale
892 \let\selectlocale\setlocale
893 \let\textlocale\setlocale
894 \let\textlanguage\setlocale
895 \let\languagegettext\setlocale

```

4.2. Errors

\@nolanerr

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

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

When the format knows about $\LaTeX 2\epsilon$, 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.

```

896 \edef\bbl@nulllanguage{\string\language=0}
897 \def\bbl@nocaption{\protect\bbl@nocaption@i}
898 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
899 \global\@namedef{#2}{\textbf{?#1?}}%
900 \@nameuse{#2}%
901 \edef\bbl@tempa{#1}%
902 \bbl@sreplace\bbl@tempa{name}}}%
903 \bbl@sreplace\bbl@tempa{NAME}}}%
904 \bbl@warning{%
905 \@backslashchar#1 not set for '\language'. Please,\\%
906 define it after the language has been loaded\\%
907 (typically in the preamble) with:\\%
908 \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
909 Feel free to contribute on github.com/latex3/babel.\\%
910 Reported}}
911 \def\bbl@tentative{\protect\bbl@tentative@i}
912 \def\bbl@tentative@i#1{%
913 \bbl@warning{%
914 Some functions for '#1' are tentative.\\%
915 They might not work as expected and their behavior\\%

```



```

916     could change in the future.\\%
917     Reported}}
918 \def\@nolanerr#1{\bbl@error{undefined-language}{#1}{}}
919 \def\@nopatterns#1{%
920   \bbl@warning
921   {No hyphenation patterns were preloaded for\\%
922     the language '#1' into the format.\\%
923     Please, configure your TeX system to add them and\\%
924     rebuild the format. Now I will use the patterns\\%
925     preloaded for \bbl@nulllanguage\space instead}}
926 \let\bbl@usehooks\@gobbletwo

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

927 \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.

```

928 \bbl@trace{Defining babelensure}
929 \newcommand\babelensure[2][]{%
930   \AddBabelHook{babel-ensure}{afterextras}{%
931     \ifcase\bbl@select@type
932       \bbl@ccl{e}%
933       \fi}%
934   \begingroup
935     \let\bbl@ens@include\@empty
936     \let\bbl@ens@exclude\@empty
937     \def\bbl@ens@fontenc{\relax}%
938     \def\bbl@tempb##1{%
939       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
940     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
941     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
942     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
943     \def\bbl@tempc{\bbl@ensure}%
944     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
945       \expandafter{\bbl@ens@include}}%
946     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
947       \expandafter{\bbl@ens@exclude}}%
948     \toks@\expandafter{\bbl@tempc}%
949     \bbl@exp{%
950   \endgroup
951   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
952 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
953 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
954   \ifx##1\undefined % 3.32 - Don't assume the macro exists
955     \edef##1{\noexpand\bbl@nocaption
956       {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}%
957   \fi
958   \ifx##1\@empty\else
959     \in@{##1}{#2}%
960     \ifin@ \else
961       \bbl@ifunset{\bbl@ensure@\language\name}%
962       {\bbl@exp{%
963         \\\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%

```

```

964         \\foreignlanguage{\language}%
965         {\ifx\relax#3\else
966         \\fontencoding{#3}\\selectfont
967         \fi
968         #####1}}}%
969     }%
970     \toks@\expandafter{##1}%
971     \edef##1{%
972         \bbl@csarg\noexpand{ensure@\language}%
973         {\the\toks@}}%
974     \fi
975     \expandafter\bbl@tempb
976     \fi}%
977     \expandafter\bbl@tempb\bbl@captionslist\today\@empty
978     \def\bbl@tempa##1{% elt for include list
979         \ifx##1\@empty\else
980             \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
981             \ifin@%
982                 \bbl@tempb##1\@empty
983             \fi
984             \expandafter\bbl@tempa
985         \fi}%
986     \bbl@tempa##1\@empty}
987     \def\bbl@captionslist{%
988         \prefacename\refname\abstractname\bibname\chaptername\appendixname
989         \contentsname\listfigurename\listtablename\indexname\figurename
990         \tablename\partname\enclname\ccname\headtoname\pagename\seename
991         \alsoname\proofname\glossaryname}

```

4.4. Short tags

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

```

992 \bbl@trace{Short tags}
993 \newcommand\babeltags[1]{%
994     \edef\bbl@tempa{\zap@space#1 \@empty}%
995     \def\bbl@tempb##1=##2\@{
996         \edef\bbl@tempc{%
997             \noexpand\newcommand
998             \expandafter\noexpand\csname ##1\endcsname{%
999                 \noexpand\protect
1000                 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1001             \noexpand\newcommand
1002             \expandafter\noexpand\csname text##1\endcsname{%
1003                 \noexpand\foreignlanguage{##2}}}
1004         \bbl@tempc}%
1005     \bbl@for\bbl@tempa\bbl@tempa{
1006         \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.

```

1007 \bbl@trace{Compatibility with language.def}
1008 \ifx\directlua\@undefined\else
1009     \ifx\bbl@luapatterns\@undefined
1010         \input luabel.def
1011     \fi
1012 \fi
1013 \ifx\bbl@languages\@undefined
1014     \ifx\directlua\@undefined
1015         \openin1 = language.def

```

```

1016 \ifeof1
1017 \closein1
1018 \message{I couldn't find the file language.def}
1019 \else
1020 \closein1
1021 \begingroup
1022 \def\addlanguage#1#2#3#4#5{%
1023 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1024 \global\expandafter\let\csname l@#1\expandafter\endcsname
1025 \csname lang@#1\endcsname
1026 \fi}%
1027 \def\uselanguage#1{%
1028 \input language.def
1029 \endgroup
1030 \fi
1031 \fi
1032 \chardef\l@english\z@
1033 \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.

```

1034 \def\addto#1#2{%
1035 \ifx#1\@undefined
1036 \def#1{#2}%
1037 \else
1038 \ifx#1\relax
1039 \def#1{#2}%
1040 \else
1041 {\toks@\expandafter{#1#2}%
1042 \xdef#1{\the\toks@}}%
1043 \fi
1044 \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.

```

1045 \bbl@trace{Hooks}
1046 \newcommand\AddBabelHook[3][{}]{%
1047 \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1048 \def\bbl@tempa#1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1049 \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1050 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1051 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1052 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1053 \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1054 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1055 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1056 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1057 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1058 \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1059 \def\bbl@elth##1{%
1060 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1061 \bbl@cs{ev@#2@#3}%
1062 \ifx\language\@undefined\else % Test required for Plain (?)
1063 \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1064 \def\bbl@elth##1{%
1065 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}%

```

```

1066 \bbl@cs{ev@#2@#1}%
1067 \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).

```

1068 \def\bbl@evargs{,% <- don't delete this comma
1069 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1070 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1071 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1072 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1073 beforestart=0,language=2,begindocument=1}
1074 \ifx\NewHook\@undefined\else % Test for Plain (?)
1075 \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1076 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1077 \fi

```

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

```

1078 \providecommand\PassOptionsToLocale[2]{%
1079 \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.

```

1080 \bbl@trace{Macros for setting language files up}
1081 \def\bbl@ldfinit{%
1082 \let\bbl@screset\@empty
1083 \let\BabelStrings\bbl@opt@string
1084 \let\BabelOptions\@empty
1085 \let\BabelLanguages\relax
1086 \ifx\originalTeX\@undefined
1087 \let\originalTeX\@empty
1088 \else
1089 \originalTeX
1090 \fi}
1091 \def\LdfInit#1#2{%
1092 \chardef\atcatcode=\catcode`\@
1093 \catcode`\@=11\relax
1094 \chardef\eqcatcode=\catcode`\=
1095 \catcode`\==12\relax
1096 \ifpackagewith{babel}{ensureinfo=off}}}%
1097 {\ifx\InputIfFileExists\@undefined\else
1098 \bbl@ifunset{\bbl@lname@#1}%
1099 {\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1100 \def\language#1}%
1101 \bbl@id@assign

```

```

1102         \bbl@load@info{#1}}}%
1103     }%
1104     \fi}%
1105     \expandafter\if\expandafter\@backslashchar
1106         \expandafter\@car\string#2\@nil
1107         \ifx#2\@undefined\else
1108             \ldf@quit{#1}%
1109         \fi
1110     \else
1111         \expandafter\ifx\csname#2\endcsname\relax\else
1112             \ldf@quit{#1}%
1113         \fi
1114     \fi
1115     \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.

```

1116 \def\ldf@quit#1{%
1117     \expandafter\main@language\expandafter{#1}%
1118     \catcode`\@=\atcatcode \let\atcatcode\relax
1119     \catcode`\==\eqcatcode \let\eqcatcode\relax
1120     \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.

```

1121 \def\bbl@afterldf{%
1122     \bbl@afterlang
1123     \let\bbl@afterlang\relax
1124     \let\BabelModifiers\relax
1125     \let\bbl@screset\relax}%
1126 \def\ldf@finish#1{%
1127     \loadlocalcfg{#1}%
1128     \bbl@afterldf
1129     \expandafter\main@language\expandafter{#1}%
1130     \catcode`\@=\atcatcode \let\atcatcode\relax
1131     \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1132 \@onlypreamble\LdfInit
1133 \@onlypreamble\ldf@quit
1134 \@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.

```

1135 \def\main@language#1{%
1136     \def\bbl@main@language{#1}%
1137     \let\language\name\bbl@main@language
1138     \let\localname\bbl@main@language
1139     \let\mainlocalname\bbl@main@language
1140     \bbl@id@assign
1141     \bbl@patterns{\language\name}}

```

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.

```

1142 \def\bbl@beforestart{%
1143   \def\nolanerr##1{%
1144     \bbl@carg\chardef{l@##1}\z@
1145     \bbl@warning{Undefined language '##1' in aux.\Reported}}%
1146   \bbl@usehooks{beforestart}{}%
1147   \global\let\bbl@beforestart\relax}
1148 \AtBeginDocument{%
1149   {\@nameuse\bbl@beforestart}}% Group!
1150   \if@files
1151     \providecommand\babel@aux[2]{}%
1152     \immediate\write\@mainaux{\unexpanded{%
1153       \providecommand\babel@aux[2]{\global\let\babel@toc@gobbletwo}}}%
1154     \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1155   \fi
1156   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1157   \ifbbl@single % must go after the line above.
1158     \renewcommand\selectlanguage[1]{}%
1159     \renewcommand\foreignlanguage[2]{#2}%
1160     \global\let\babel@aux@gobbletwo % Also as flag
1161   \fi}
1162 %
1163 \ifcase\bbl@engine\or
1164   \AtBeginDocument{\pagedir\bodydir}
1165 \fi

```

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

```

1166 \def\select@language@x#1{%
1167   \ifcase\bbl@select@type
1168     \bbl@ifsamestring\language#1{\select@language{#1}}%
1169   \else
1170     \select@language{#1}%
1171   \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.

```

1172 \bbl@trace{Shorhands}
1173 \def\bbl@withactive#1#2{%
1174   \begingroup
1175     \lccode`~=#2\relax
1176     \lowercase{\endgroup#1~}}

```

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

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

```

1177 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1178   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1179   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1180   \ifx\nfss@catcodes\undefined\else
1181     \begingroup
1182       \catcode`#1\active
1183       \nfss@catcodes
1184       \ifnum\catcode`#1=\active
1185         \endgroup
1186       \bbl@add\nfss@catcodes{\@makeother#1}%
1187     \else

```

```

1188         \endgroup
1189     \fi
1190 \fi}

```

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

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

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

```

1191 \def\bbl@active@def#1#2#3#4{%
1192   \@namedef{#3#1}{%
1193     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1194       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1195     \else
1196       \bbl@afterfi\csname#2@sh@#1@\endcsname
1197   \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.

```

1198   \long\@namedef{#3@arg#1}##1{%
1199     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1200       \bbl@afterelse\csname#4#1\endcsname##1%
1201     \else
1202       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1203   \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.

```

1204 \def\initiate@active@char#1{%
1205   \bbl@ifunset{active@char\string#1}%
1206   {\bbl@withactive
1207     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1208   {}}

```

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).

```

1209 \def\@initiate@active@char#1#2#3{%
1210   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1211   \ifx#1@\undefined
1212     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1@\undefined}}%
1213   \else
1214     \bbl@csarg\let{oridef@#2}#1%
1215     \bbl@csarg\edef{oridef@#2}{%
1216       \let\noexpand#1%
1217       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1218   \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*").

```

1219 \ifx#1#3\relax
1220 \expandafter\let\csname normal@char#2\endcsname#3%
1221 \else
1222 \bbl@info{Making #2 an active character}%
1223 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1224 \@namedef{normal@char#2}{%
1225 \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1226 \else
1227 \@namedef{normal@char#2}{#3}%
1228 \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).

```

1229 \bbl@restoreactive{#2}%
1230 \AtBeginDocument{%
1231 \catcode`#2\active
1232 \if@files
1233 \immediate\write\@mainaux{\catcode`\string#2\active}%
1234 \fi}%
1235 \expandafter\bbl@add@special\csname#2\endcsname
1236 \catcode`#2\active
1237 \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}).

```

1238 \let\bbl@tempa\@firstoftwo
1239 \if\string^#2%
1240 \def\bbl@tempa{\noexpand\textormath}%
1241 \else
1242 \ifx\bbl@mathnormal\@undefined\else
1243 \let\bbl@tempa\bbl@mathnormal
1244 \fi
1245 \fi
1246 \expandafter\edef\csname active@char#2\endcsname{%
1247 \bbl@tempa
1248 {\noexpand\if@safe@actives
1249 \noexpand\expandafter
1250 \expandafter\noexpand\csname normal@char#2\endcsname
1251 \noexpand\else
1252 \noexpand\expandafter
1253 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1254 \noexpand\fi}%
1255 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1256 \bbl@csarg\edef{doactive#2}{%
1257 \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

\active@prefix{char} \normal@char{char}

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

```

1258 \bbl@csarg\edef{active@#2}{%
1259 \noexpand\active@prefix\noexpand#1%

```



```

1260 \expandafter\noexpand\csname active@char#2\endcsname}%
1261 \bbl@csarg\edef{normal@#2}{%
1262 \noexpand\active@prefix\noexpand#1%
1263 \expandafter\noexpand\csname normal@char#2\endcsname}%
1264 \bbl@ncarg\let#1\bbl@normal@#2}%

```

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

```

1265 \bbl@active@def#2\user@group{user@active}{language@active}%
1266 \bbl@active@def#2\language@group{language@active}{system@active}%
1267 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1268 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1269 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1270 \expandafter\edef\csname\user@group @sh#2@string\protect\endcsname
1271 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1272 \if\string'#2%
1273 \let\prim@s\bbl@prim@s
1274 \let\active@math@prime#1%
1275 \fi
1276 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1277 << *More package options >> ≡
1278 \DeclareOption{math=active}{}
1279 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1280 << /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*.

```

1281 \ifpackagewith{babel}{KeepShorthandsActive}%
1282 {\let\bbl@restoreactive@gobble}%
1283 {\def\bbl@restoreactive#1{%
1284 \bbl@exp{%
1285 \\\AfterBabelLanguage\\CurrentOption
1286 {\catcode`#1=\the\catcode`#1\relax}%
1287 \\\AtEndOfPackage
1288 {\catcode`#1=\the\catcode`#1\relax}}}%
1289 \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.

```

1290 \def\bbl@sh@select#1#2{%
1291 \expandafter\ifx\csname#1@sh#2@sel\endcsname\relax
1292 \bbl@afterelse\bbl@scndcs
1293 \else
1294 \bbl@afterfi\csname#1@sh#2@sel\endcsname
1295 \fi}

```

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

```

1296 \begingroup
1297 \bbl@ifunset{ifincsname}
1298 {\gdef\active@prefix#1{%
1299   \ifx\protect\@typeset@protect
1300   \else
1301     \ifx\protect\@unexpandable@protect
1302       \noexpand#1%
1303     \else
1304       \protect#1%
1305     \fi
1306     \expandafter\@gobble
1307   \fi}}
1308 {\gdef\active@prefix#1{%
1309   \ifincsname
1310     \string#1%
1311     \expandafter\@gobble
1312   \else
1313     \ifx\protect\@typeset@protect
1314     \else
1315       \ifx\protect\@unexpandable@protect
1316         \noexpand#1%
1317       \else
1318         \protect#1%
1319       \fi
1320       \expandafter\expandafter\expandafter\@gobble
1321     \fi
1322   \fi}}
1323 \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).

```

1324 \newif\if@safe@actives
1325 \@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.

```

1326 \def\bbl@restore@actives{\if@safe@actives\@safe@activessfalse\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.

```

1327 \chardef\bbl@activated\z@
1328 \def\bbl@activate#1{%
1329   \chardef\bbl@activated\@ne
1330   \bbl@withactive{\expandafter\let\expandafter}#1%
1331   \csname bbl@active@\string#1\endcsname}
1332 \def\bbl@deactivate#1{%
1333   \chardef\bbl@activated\tw@
1334   \bbl@withactive{\expandafter\let\expandafter}#1%

```

```
1335 \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs

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

```
1336 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1337 \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.

```
1338 \def\babel@texpdf#1#2#3#4{%
1339 \ifx\texorpdfstring\undefined
1340 \textormath{#1}{#3}%
1341 \else
1342 \texorpdfstring{\textormath{#1}{#3}}{#2}%
1343 % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1344 \fi}
1345 %
1346 \def\declare@shorthand#1#2{\@decl@short{#1}#2@nil}
1347 \def\@decl@short#1#2#3@nil#4{%
1348 \def\bbl@tempa{#3}%
1349 \ifx\bbl@tempa\empty
1350 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1351 \bbl@ifunset{#1@sh@\string#2@}{}%
1352 {\def\bbl@tempa{#4}%
1353 \expandafter\ifx\csname#1@sh@\string#2@endcsname\bbl@tempa
1354 \else
1355 \bbl@info
1356 {Redefining #1 shorthand \string#2\}%
1357 in language \CurrentOption}%
1358 \fi}%
1359 \@namedef{#1@sh@\string#2@}{#4}%
1360 \else
1361 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1362 \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1363 {\def\bbl@tempa{#4}%
1364 \expandafter\ifx\csname#1@sh@\string#2@\string#3@endcsname\bbl@tempa
1365 \else
1366 \bbl@info
1367 {Redefining #1 shorthand \string#2\string#3\}%
1368 in language \CurrentOption}%
1369 \fi}%
1370 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1371 \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.

```
1372 \def\textormath{%
1373 \ifmmode
1374 \expandafter\@secondoftwo
1375 \else
1376 \expandafter\@firstoftwo
1377 \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’.

```
1378 \def\user@group{user}
1379 \def\language@group{english}
1380 \def\system@group{system}
```

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

```
1381 \def\usesshorthands{%
1382   \ifstar\bbbl@usesesh@s{\bbbl@usesesh@x{}}
1383   \def\bbbl@usesesh@s#1{%
1384     \bbbl@usesesh@x
1385     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbbl@activate{#1}}}%
1386     {#1}}
1387   \def\bbbl@usesesh@x#1#2{%
1388     \bbbl@ifshorthand{#2}%
1389     {\def\user@group{user}%
1390      \initiate@active@char{#2}%
1391      #1%
1392      \bbbl@activate{#2}}%
1393     {\bbbl@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 \bbbl@set@user@generic); we make also sure {} and \protect are taken into account in this new top level.

```
1394 \def\user@language@group{user@\language@group}
1395 \def\bbbl@set@user@generic#1#2{%
1396   \bbbl@ifunset{user@generic@active#1}%
1397   {\bbbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1398    \bbbl@active@def#1\user@group{user@generic@active}{language@active}%
1399    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1400      \expandafter\noexpand\csname normal@char#1\endcsname}%
1401    \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1402      \expandafter\noexpand\csname user@active#1\endcsname}}%
1403   \@empty}
1404 \newcommand\defineshorthand[3][user]{%
1405   \edef\bbbl@tempa{\zap@space#1 \@empty}%
1406   \bbbl@for\bbbl@tempb\bbbl@tempa{%
1407     \if*\expandafter\@car\bbbl@tempb\@nil
1408       \edef\bbbl@tempb{user@\expandafter\@gobble\bbbl@tempb}%
1409       \@expandtwoargs
1410       \bbbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbbl@tempb
1411     \fi
1412     \declare@shorthand{\bbbl@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.

```
1413 \def\languageshorthands#1{%
1414   \bbbl@ifsamestring{none}{#1}{}%
1415   \bbbl@once{short-\localename-#1}{%
1416     \bbbl@info{'\localename' activates '#1' shorthands.\@Reported}}%
1417   \def\language@group{#1}}
```

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

```

1418 \def\aliasshorthand#1#2{%
1419   \bbl@ifshorthand{#2}%
1420   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1421     \ifx\document\@notprerr
1422       \@notshorthand{#2}%
1423     \else
1424       \initiate@active@char{#2}%
1425       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1426       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1427       \bbl@activate{#2}%
1428     \fi
1429   \fi}%
1430   {\bbl@error{shorthand-is-off}{#2}{}}}
```

\@notshorthand

```

1431 \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.

```

1432 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1433 \DeclareRobustCommand*\shorthandoff{%
1434   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1435 \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.

```

1436 \def\bbl@switch@sh#1#2{%
1437   \ifx#2\@nnil\else
1438     \bbl@ifunset{bbl@active@\string#2}%
1439     {\bbl@error{not-a-shorthand-b}{#2}{}}%
1440     {\ifcase#1%   off, on, off*
1441       \catcode`#2\relax
1442     \or
1443       \catcode`#2\active
1444       \bbl@ifunset{bbl@shdef@\string#2}%
1445       {}%
1446       {\bbl@withactive{\expandafter\let\expandafter}#2%
1447         \csname bbl@shdef@\string#2\endcsname
1448         \bbl@csarg\let{shdef@\string#2}\relax}%
1449       \ifcase\bbl@activated\or
1450         \bbl@activate{#2}%
1451       \else
1452         \bbl@deactivate{#2}%
1453       \fi
1454     \or
1455       \bbl@ifunset{bbl@shdef@\string#2}%
1456       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1457       {}%
1458       \csname bbl@oricat@\string#2\endcsname
1459       \csname bbl@oridef@\string#2\endcsname
1460     \fi}%
```

```

1461 \bbl@afterfi\bbl@switch@sh#1%
1462 \fi}

```

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

```

1463 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1464 \def\bbl@putsh#1{%
1465 \bbl@ifunset{\bbl@active@\string#1}%
1466 {\bbl@putsh@i#1\@empty\@nnil}%
1467 {\csname bbl@active@\string#1\endcsname}}
1468 \def\bbl@putsh@i#1#2\@nnil{%
1469 \csname\language@group @sh@\string#1@%
1470 \ifx\@empty#2\else\string#2@\fi\endcsname}
1471 %
1472 \ifx\bbl@opt@shorthands\@nnil\else
1473 \let\bbl@s@initiate@active@char@initiate@active@char
1474 \def\initiate@active@char#1{%
1475 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1476 \let\bbl@s@switch@sh\bbl@switch@sh
1477 \def\bbl@switch@sh#1#2{%
1478 \ifx#2\@nnil\else
1479 \bbl@afterfi
1480 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1481 \fi}
1482 \let\bbl@s@activate\bbl@activate
1483 \def\bbl@activate#1{%
1484 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1485 \let\bbl@s@deactivate\bbl@deactivate
1486 \def\bbl@deactivate#1{%
1487 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1488 \fi

```

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

```

1489 \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.

```

1490 \def\bbl@prim@s{%
1491 \prime\futurelet\@let@token\bbl@pr@m@s}
1492 \def\bbl@if@primes#1#2{%
1493 \ifx#1\@let@token
1494 \expandafter\@firstoftwo
1495 \else\ifx#2\@let@token
1496 \bbl@afterelse\expandafter\@firstoftwo
1497 \else
1498 \bbl@afterfi\expandafter\@secondoftwo
1499 \fi\fi}
1500 \begingroup
1501 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1502 \catcode`\'=12 \catcode`\\"=\active \lccode`\\"=\^
1503 \lowercase{%
1504 \gdef\bbl@pr@m@s{%
1505 \bbl@if@primes" '%
1506 \pr@@@s
1507 {\bbl@if@primes*\^{\pr@@@t\egroup}}}
1508 \endgroup

```

Usually the ~ is active and expands to \penalty\M\L. When it is written to the aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it

is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

```
1509 \initiate@active@char{~}
1510 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1511 \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.

```
1512 \expandafter\def\csname OT1dqpos\endcsname{127}
1513 \expandafter\def\csname T1dqpos\endcsname{4}
```

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

```
1514 \ifx\f@encoding\undefined
1515   \def\f@encoding{OT1}
1516 \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.

```
1517 \bbl@trace{Language attributes}
1518 \newcommand\languageattribute[2]{%
1519   \def\bbl@tempc{#1}%
1520   \bbl@fixname\bbl@tempc
1521   \bbl@iflanguage\bbl@tempc{%
1522     \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.

```
1523     \ifx\bbl@known@attrs\undefined
1524       \in@false
1525     \else
1526       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1527     \fi
1528     \ifin@
1529       \bbl@warning{%
1530         You have more than once selected the attribute '##1'\%
1531         for language #1. Reported}%
1532     \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 T_EX-code.

```
1533       \bbl@info{Activated '##1' attribute for\\%
1534         '\bbl@tempc'. Reported}%
1535       \bbl@exp{%
1536         \\ \bbl@add@list\\ \bbl@known@attrs{\bbl@tempc-##1}}%
1537       \edef\bbl@tempa{\bbl@tempc-##1}%
1538       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1539       {\csname\bbl@tempc_attr@##1\endcsname}%
1540       {\@attrerr{\bbl@tempc}{##1}}%
1541     \fi}}
1542 \onlypreamble\languageattribute
```

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

```
1543 \newcommand*{\@attrerr}[2]{%
1544   \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}`.

```
1545 \def\bbl@declare@ttribute#1#2#3{%
1546   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1547   \ifin@
1548     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1549   \fi
1550   \bbl@add@list\bbl@attributes{#1-#2}%
1551   \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.

```
1552 \def\bbl@ifattributeset#1#2#3#4{%
1553   \ifx\bbl@known@attribs\@undefined
1554     \in@false
1555   \else
1556     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1557   \fi
1558   \ifin@
1559     \bbl@afterelse#3%
1560   \else
1561     \bbl@afterfi#4%
1562   \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.

```
1563 \def\bbl@ifknown@ttrib#1#2{%
1564   \let\bbl@tempa\@secondoftwo
1565   \bbl@loopx\bbl@tempb{#2}{%
1566     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1567   \ifin@
1568     \let\bbl@tempa\@firstoftwo
1569   \else
1570     \fi}%
1571   \bbl@tempa}
```

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

```
1572 \def\bbl@clear@ttribs{%
1573   \ifx\bbl@attributes\@undefined\else
1574     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1575       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1576     \let\bbl@attributes\@undefined
1577   \fi}
1578 \def\bbl@clear@ttrib#1-#2.{%
1579   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1580 \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.

```
1581 \bbl@trace{Macros for saving definitions}
1582 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1583 \newcount\babel@savecnt
1584 \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.

```
1585 \def\babel@save#1{%
1586   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1587   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1588     \expandafter{\expandafter,\bbl@savedextras,}}%
1589   \expandafter\in@\bbl@tempa
1590   \ifin@else
1591     \bbl@add\bbl@savedextras{,{#1,}}%
1592     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1593     \toks@\expandafter{\originalTeX\let#1=}%
1594     \bbl@exp{%
1595       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1596     \advance\babel@savecnt\@ne
1597   \fi}
1598 \def\babel@savevariable#1{%
1599   \toks@\expandafter{\originalTeX #1=}%
1600   \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`.

```
1601 \def\bbl@redefine#1{%
1602   \edef\bbl@tempa{\bbl@stripslash#1}%
1603   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1604   \expandafter\def\csname\bbl@tempa\endcsname}
1605 \@onlypreamble\bbl@redefine
```

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

```
1606 \def\bbl@redefine@long#1{%
1607   \edef\bbl@tempa{\bbl@stripslash#1}%
1608   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1609   \long\expandafter\def\csname\bbl@tempa\endcsname}
1610 \@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`.

```
1611 \def\bbl@redefineroobust#1{%
1612   \edef\bbl@tempa{\bbl@stripslash#1}%
1613   \bbl@iifunset{\bbl@tempa\space}%
1614   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1615    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1616   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1617   \@namedef{\bbl@tempa\space}}
1618 \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.

```
1619 \def\bbl@frenchspacing{%
1620   \ifnum\the\sfcode`\.\=\m
1621     \let\bbl@nonfrenchspacing\relax
1622   \else
1623     \frenchspacing
1624     \let\bbl@nonfrenchspacing\nonfrenchspacing
1625   \fi}
1626 \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.

```
1627 \let\bbl@elt\relax
1628 \edef\bbl@fs@chars{%
1629   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1630   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1631   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1632 \def\bbl@pre@fs{%
1633   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1634   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1635 \def\bbl@post@fs{%
1636   \bbl@save@sfcodes
1637   \edef\bbl@tempa{\bbl@cl{frspc}}%
1638   \edef\bbl@tempa{\expandafter\car\bbl@tempa\@nil}%
1639   \if u\bbl@tempa      % do nothing
1640   \else\if n\bbl@tempa % non french
1641     \def\bbl@elt##1##2##3{%
1642       \ifnum\sfcode`##1=##2\relax
1643       \babel@savevariable{\sfcode`##1}%
1644       \sfcode`##1=##3\relax
1645     \fi}%
1646     \bbl@fs@chars
1647   \else\if y\bbl@tempa % french
1648     \def\bbl@elt##1##2##3{%
1649       \ifnum\sfcode`##1=##3\relax
1650       \babel@savevariable{\sfcode`##1}%
1651       \sfcode`##1=##2\relax
1652     \fi}%
1653     \bbl@fs@chars
1654   \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

```

1655 \bbl@trace{Hyphens}
1656 \@onlypreamble\babelhyphenation
1657 \AtEndOfPackage{%
1658   \newcommand\babelhyphenation[2][\@empty]{%
1659     \ifx\bbl@hyphenation@\relax
1660       \let\bbl@hyphenation@\@empty
1661     \fi
1662     \ifx\bbl@hyphlist\@empty\else
1663       \bbl@warning{%
1664         You must not intermingle \string\selectlanguage\space and\\%
1665         \string\babelhyphenation\space or some exceptions will not\\%
1666         be taken into account. Reported}%
1667     \fi
1668     \ifx\@empty#1%
1669       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1670     \else
1671       \bbl@vforeach{#1}{%
1672         \def\bbl@tempa{##1}%
1673         \bbl@fixname\bbl@tempa
1674         \bbl@iflanguage\bbl@tempa{%
1675           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1676             \bbl@ifunset{\bbl@hyphenation@\bbl@tempa}%
1677             {}%
1678             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1679             #2}}}%
1680       \fi}}

```

```

1681 \ifx\NewDocumentCommand\undefined\else
1682   \NewDocumentCommand\babelhyphenmins{sommo}{%
1683     \IfNoValueTF{#2}%
1684       {\protected@edef\bbl@hyphenmins@{\set@hyphenmins{#3}{#4}}}%
1685       \IfValueT{#5}{%
1686         \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1687       \IfBooleanT{#1}{%
1688         \lefthyphenmin=#3\relax
1689         \righthyphenmin=#4\relax
1690         \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1691   {\edef\bbl@tempb{\zap@space#2 \@empty}%
1692     \bbl@for\bbl@tempa\bbl@tempb{%
1693       \@namedef{bbl@hyphenmins@\bbl@tempa}{\set@hyphenmins{#3}{#4}}}%
1694     \IfValueT{#5}{%
1695       \@namedef{bbl@hyphenatmin@\bbl@tempa}{\hyphenationmin=#5\relax}}}%
1696     \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}}{}}}%
1697 \fi

```

```

1698 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1699 \def\bbl@t@one{T1}
1700 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

```

1701 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1702 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1703 \def\bbl@hyphen{%

```

```

1704 \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1705 \def\bbl@hyphen@i#1#2{%
1706 \lowercase{\bbl@ifunset{\bbl@hy#1#2\@empty}}}%
1707 {\csname bbl@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1708 {\lowercase{\csname bbl@hy#1#2\@empty\endcsname}}}

```

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

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

```

1709 \def\bbl@usehyphen#1{%
1710 \leavevmode
1711 \ifdim\lastskip>z@\mbox{#1}\else\nobreak#1\fi
1712 \nobreak\hskip\z@skip}
1713 \def\bbl@usehyphen#1{%
1714 \leavevmode\ifdim\lastskip>z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1715 \def\bbl@hyphenchar{%
1716 \ifnum\hyphenchar\font=\m@ne
1717 \babe\nullhyphen
1718 \else
1719 \char\hyphenchar\font
1720 \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.

```

1721 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1722 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1723 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1724 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1725 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1726 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1727 \def\bbl@hy@repeat{%
1728 \bbl@usehyphen{%
1729 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1730 \def\bbl@hy@repeat{%
1731 \bbl@usehyphen{%
1732 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1733 \def\bbl@hy@empty{\hskip\z@skip}
1734 \def\bbl@hy@empty{\discretionary{}{}{}}

```

\bbl@disc For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.

```

1735 \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.

```

1736 \bbl@trace{Multiencoding strings}
1737 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1738 <<More package options>> ≡
1739 \DeclareOption{nocase}{}
1740 <</More package options>>

```

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

```

1741 <<{*More package options}>> ≡
1742 \let\bbl@opt@strings\@nnil % accept strings=value
1743 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1744 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1745 \def\BabelStringsDefault{generic}
1746 <</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.

```

1747 \@onlypreamble\StartBabelCommands
1748 \def\StartBabelCommands{%
1749   \begingroup
1750   \@tempcnta="7F
1751   \def\bbl@tempa{%
1752     \ifnum\@tempcnta>"FF\else
1753       \catcode\@tempcnta=11
1754       \advance\@tempcnta\@ne
1755       \expandafter\bbl@tempa
1756     \fi}%
1757   \bbl@tempa
1758   <@Macros local to BabelCommands@>
1759   \def\bbl@provstring##1##2{%
1760     \providecommand##1{##2}%
1761     \bbl@tglobal##1}%
1762   \global\let\bbl@scafter\@empty
1763   \let\StartBabelCommands\bbl@startcmds
1764   \ifx\BabelLanguages\relax
1765     \let\BabelLanguages\CurrentOption
1766   \fi
1767   \begingroup
1768   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1769   \StartBabelCommands}
1770 \def\bbl@startcmds{%
1771   \ifx\bbl@screset\@nnil\else
1772     \bbl@usehooks{stopcommands}{}%
1773   \fi
1774   \endgroup
1775   \begingroup
1776   \@ifstar
1777   {\ifx\bbl@opt@strings\@nnil
1778     \let\bbl@opt@strings\BabelStringsDefault
1779     \fi
1780     \bbl@startcmds@i}%
1781   \bbl@startcmds@i}
1782 \def\bbl@startcmds@i#1#2{%
1783   \edef\bbl@L{\zap@space#1 \@empty}%
1784   \edef\bbl@G{\zap@space#2 \@empty}%
1785   \bbl@startcmds@ii}
1786 \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.

```

1787 \newcommand\bbl@startcmds@ii[1][\@empty]{%

```

```

1788 \let\SetString\@gobbletwo
1789 \let\bbl@stringdef\@gobbletwo
1790 \let\AfterBabelCommands\@gobble
1791 \ifx\@empty#1%
1792   \def\bbl@sc@label{generic}%
1793   \def\bbl@encstring##1##2{%
1794     \ProvideTextCommandDefault##1{##2}%
1795     \bbl@tglobal##1%
1796     \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1797   \let\bbl@sctest\in@true
1798 \else
1799   \let\bbl@sc@charset\space % <- zapped below
1800   \let\bbl@sc@fontenc\space % <- " "
1801   \def\bbl@tempa##1=##2\@nil{%
1802     \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1803   \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1804   \def\bbl@tempa##1 ##2{% space -> comma
1805     ##1%
1806     \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1807   \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1808   \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1809   \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1810   \def\bbl@encstring##1##2{%
1811     \bbl@foreach\bbl@sc@fontenc{%
1812       \bbl@ifunset{T@###1}%
1813       {}%
1814       {\ProvideTextCommand##1{###1}{##2}%
1815         \bbl@tglobal##1%
1816         \expandafter
1817         \bbl@tglobal\csname###1\string##1\endcsname}}}%
1818   \def\bbl@sctest{%
1819     \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
1820 \fi
1821 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1822 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1823   \let\AfterBabelCommands\bbl@aftercmds
1824   \let\SetString\bbl@setstring
1825   \let\bbl@stringdef\bbl@encstring
1826 \else % i.e., strings=value
1827   \bbl@sctest
1828 \fin@
1829   \let\AfterBabelCommands\bbl@aftercmds
1830   \let\SetString\bbl@setstring
1831   \let\bbl@stringdef\bbl@provstring
1832 \fi\fi\fi
1833 \bbl@scswitch
1834 \ifx\bbl@G\@empty
1835   \def\SetString##1##2{%
1836     \bbl@error{missing-group}{##1}{}}%
1837 \fi
1838 \ifx\@empty#1%
1839   \bbl@usehooks{defaultcommands}{}%
1840 \else
1841   \@expandtwoargs
1842   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1843 \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).

```

1844 \def\bbl@forlang#1#2{%
1845   \bbl@for#1\bbl@L{%
1846     \bbl@xin@{,#1,},{,\BabelLanguages,}%
1847     \ifin@#2\relax\fi}}
1848 \def\bbl@scswitch{%
1849   \bbl@forlang\bbl@tempa{%
1850     \ifx\bbl@G\@empty\else
1851       \ifx\SetString@gobbletwo\else
1852         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1853         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1854         \ifin@\else
1855           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1856           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1857         \fi
1858       \fi
1859     \fi}}
1860 \AtEndOfPackage{%
1861   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1862   \let\bbl@scswitch\relax}
1863 \@onlypreamble\EndBabelCommands
1864 \def\EndBabelCommands{%
1865   \bbl@usehooks{stopcommands}{}%
1866   \endgroup
1867   \endgroup
1868   \bbl@scafter}
1869 \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.

```

1870 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1871   \bbl@forlang\bbl@tempa{%
1872     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1873     \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1874     {\bbl@exp{%
1875       \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1876     }%
1877     \def\BabelString{#2}%
1878     \bbl@usehooks{stringprocess}{}%
1879     \expandafter\bbl@stringdef
1880     \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.

```

1881 \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.

```

1882 <<*Macros local to BabelCommands>> ≡
1883 \def\SetStringLoop##1#2{%
1884   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1885   \count@\z@
1886   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1887     \advance\count@\@ne
1888     \toks@\expandafter{\bbl@tempa}%
1889     \bbl@exp{%
1890       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%

```

```

1891 \count@=\the\count@\relax}}}%
1892 <</Macros local to BabelCommands>>

```

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

```

1893 \def\bbl@aftercmds#1{%
1894 \toks@\expandafter{\bbl@scafter#1}%
1895 \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.

```

1896 <<*Macros local to BabelCommands>> ≡
1897 \newcommand\SetCase[3][]{%
1898 \def\bbl@tempa####1####2{%
1899 \ifx####1\@empty\else
1900 \bbl@carg\bbl@add{extras\CurrentOption}{%
1901 \bbl@carg\babel@save{c__text_uppercase_\string####1_tl}%
1902 \bbl@carg\def{c__text_uppercase_\string####1_tl}{####2}%
1903 \bbl@carg\babel@save{c__text_lowercase_\string####2_tl}%
1904 \bbl@carg\def{c__text_lowercase_\string####2_tl}{####1}}%
1905 \expandafter\bbl@tempa
1906 \fi}%
1907 \bbl@tempa##1\@empty\@empty
1908 \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1909 <</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.

```

1910 <<*Macros local to BabelCommands>> ≡
1911 \newcommand\SetHyphenMap[1]{%
1912 \bbl@forlang\bbl@tempa{%
1913 \expandafter\bbl@stringdef
1914 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1915 <</Macros local to BabelCommands>>

```

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

```

1916 \newcommand\BabelLower[2]{% one to one.
1917 \ifnum\lccode#1=#2\else
1918 \babel@savevariable{\lccode#1}%
1919 \lccode#1=#2\relax
1920 \fi}
1921 \newcommand\BabelLowerMM[4]{% many-to-many
1922 \@tempcnta=#1\relax
1923 \@tempcntb=#4\relax
1924 \def\bbl@tempa{%
1925 \ifnum\@tempcnta>#2\else
1926 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1927 \advance\@tempcnta#3\relax
1928 \advance\@tempcntb#3\relax
1929 \expandafter\bbl@tempa
1930 \fi}%
1931 \bbl@tempa}
1932 \newcommand\BabelLowerM0[4]{% many-to-one
1933 \@tempcnta=#1\relax
1934 \def\bbl@tempa{%
1935 \ifnum\@tempcnta>#2\else
1936 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1937 \advance\@tempcnta#3
1938 \expandafter\bbl@tempa
1939 \fi}%
1940 \bbl@tempa}

```


The following package options control the behavior of hyphenation mapping.

```

1941 <<*More package options>> ≡
1942 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1943 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1944 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1945 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1946 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1947 <</More package options>>

```

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

```

1948 \AtEndOfPackage{%
1949   \ifx\bbl@opt@hyphenmap\undefined
1950     \bbl@xin@{,}{\bbl@language@opts}%
1951     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1952   \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.

```

1953 \newcommand\setlocalecaption{%
1954   \ifstar\bbl@setcaption@s\bbl@setcaption@x}
1955 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1956   \bbl@trim@def\bbl@tempa{#2}%
1957   \bbl@xin@{.template}{\bbl@tempa}%
1958   \ifin@
1959     \bbl@ini@captions@template{#3}{#1}%
1960   \else
1961     \edef\bbl@tempd{%
1962       \expandafter\expandafter\expandafter
1963       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1964     \bbl@xin@
1965       {\expandafter\string\csname #2name\endcsname}%
1966       {\bbl@tempd}%
1967     \ifin@ % Renew caption
1968       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
1969     \ifin@
1970       \bbl@exp{%
1971         \\bbl@ifsamestring{\bbl@tempa}{\language name}%
1972         {\bbl@scset\<#2name>\<#1#2name>}%
1973         {}}%
1974       \else % Old way converts to new way
1975         \bbl@ifunset{#1#2name}%
1976         {\bbl@exp{%
1977           \\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1978           \\bbl@ifsamestring{\bbl@tempa}{\language name}%
1979           {\def\<#2name>\<#1#2name>}}%
1980           {}}}%
1981       {}%
1982     \fi
1983   \else
1984     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1985     \ifin@ % New way
1986     \bbl@exp{%
1987       \\bbl@add\<captions#1>\bbl@scset\<#2name>\<#1#2name>}%
1988       \\bbl@ifsamestring{\bbl@tempa}{\language name}%
1989       {\bbl@scset\<#2name>\<#1#2name>}%
1990       {}}%
1991     \else % Old way, but defined in the new way
1992     \bbl@exp{%
1993       \\bbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1994       \\bbl@ifsamestring{\bbl@tempa}{\language name}%

```

```

1995      {\def\<#2name>{\<#1#2name>}}%
1996      {}}%
1997      \fi%
1998      \fi
1999      \@namedef{#1#2name}{#3}%
2000      \toks@{\expandafter{\bbl@captionslist}%
2001      \bbl@exp{\in{\<#2name>}{\the\toks@}}}%
2002      \ifin@else
2003      \bbl@exp{\bbl@add{\bbl@captionslist{\<#2name>}}}%
2004      \bbl@tglobal\bbl@captionslist
2005      \fi
2006      \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.

```

2007 \bbl@trace{Macros related to glyphs}
2008 \def\set@low@box#1{\setbox\tw@{\hbox{,}}\setbox\z@{\hbox{#1}}%
2009   \dimen\z@{\ht\z@ \advance\dimen\z@ -\ht\tw@}%
2010   \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.

```

2011 \def\save@sf@q#1{\leavevmode
2012   \begingroup
2013   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2014   \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.

```

2015 \ProvideTextCommand{\quotedblbase}{OT1}{%
2016   \save@sf@q{\set@low@box{\textquotedblright}/}%
2017   \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.

```

2018 \ProvideTextCommandDefault{\quotedblbase}{%
2019   \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2020 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2021   \save@sf@q{\set@low@box{\textquoteright}/}%
2022   \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.

```

2023 \ProvideTextCommandDefault{\quotesinglbase}{%
2024   \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

2025 \ProvideTextCommand{\guillemetleft}{OT1}{%
2026   \ifmmode
2027     \ll
2028   \else
2029     \save@sf@q{\nobreak
2030       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2031   \fi}
2032 \ProvideTextCommand{\guillemetright}{OT1}{%
2033   \ifmmode
2034     \gg
2035   \else
2036     \save@sf@q{\nobreak
2037       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2038   \fi}
2039 \ProvideTextCommand{\guillemotleft}{OT1}{%
2040   \ifmmode
2041     \ll
2042   \else
2043     \save@sf@q{\nobreak
2044       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2045   \fi}
2046 \ProvideTextCommand{\guillemotright}{OT1}{%
2047   \ifmmode
2048     \gg
2049   \else
2050     \save@sf@q{\nobreak
2051       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2052   \fi}

```

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

```

2053 \ProvideTextCommandDefault{\guillemetleft}{%
2054   \UseTextSymbol{OT1}{\guillemetleft}}
2055 \ProvideTextCommandDefault{\guillemetright}{%
2056   \UseTextSymbol{OT1}{\guillemetright}}
2057 \ProvideTextCommandDefault{\guillemotleft}{%
2058   \UseTextSymbol{OT1}{\guillemotleft}}
2059 \ProvideTextCommandDefault{\guillemotright}{%
2060   \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2061 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2062   \ifmmode
2063     <%
2064   \else
2065     \save@sf@q{\nobreak
2066       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2067   \fi}
2068 \ProvideTextCommand{\guilsinglright}{OT1}{%
2069   \ifmmode
2070     >%
2071   \else
2072     \save@sf@q{\nobreak
2073       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2074   \fi}

```

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

```

2075 \ProvideTextCommandDefault{\guilsinglleft}{%
2076   \UseTextSymbol{OT1}{\guilsinglleft}}
2077 \ProvideTextCommandDefault{\guilsinglright}{%
2078   \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.

```
2079 \DeclareTextCommand{\ij}{OT1}{%
2080   i\kern-0.02em\bbl@allowhyphens j}
2081 \DeclareTextCommand{\IJ}{OT1}{%
2082   I\kern-0.02em\bbl@allowhyphens J}
2083 \DeclareTextCommand{\ij}{T1}{\char188}
2084 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2085 \ProvideTextCommandDefault{\ij}{%
2086   \UseTextSymbol{OT1}{\ij}}
2087 \ProvideTextCommandDefault{\IJ}{%
2088   \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).

```
2089 \def\crrtic@{\hrule height0.1ex width0.3em}
2090 \def\crrtic@{\hrule height0.1ex width0.33em}
2091 \def\ddj@{%
2092   \setbox0\hbox{d}\dimen@=\ht0
2093   \advance\dimen@lex
2094   \dimen@.45\dimen@
2095   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2096   \advance\dimen@ii.5ex
2097   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2098 \def\DDJ@{%
2099   \setbox0\hbox{D}\dimen@=.55\ht0
2100   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2101   \advance\dimen@ii.15ex % correction for the dash position
2102   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2103   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2104   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2105 %
2106 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2107 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2108 \ProvideTextCommandDefault{\dj}{%
2109   \UseTextSymbol{OT1}{\dj}}
2110 \ProvideTextCommandDefault{\DJ}{%
2111   \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.

```
2112 \DeclareTextCommand{\SS}{OT1}{SS}
2113 \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.

```
2114 \ProvideTextCommandDefault{\glq}{%  
2115 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2116 \ProvideTextCommand{\grq}{T1}{%  
2117 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}  
2118 \ProvideTextCommand{\grq}{TU}{%  
2119 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}  
2120 \ProvideTextCommand{\grq}{OT1}{%  
2121 \save@sf@q{\kern-.0125em  
2122 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%  
2123 \kern.07em\relax}}  
2124 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq

\grqq The ‘german’ double quotes.

```
2125 \ProvideTextCommandDefault{\glqq}{%  
2126 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2127 \ProvideTextCommand{\grqq}{T1}{%  
2128 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}  
2129 \ProvideTextCommand{\grqq}{TU}{%  
2130 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}  
2131 \ProvideTextCommand{\grqq}{OT1}{%  
2132 \save@sf@q{\kern-.07em  
2133 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%  
2134 \kern.07em\relax}}  
2135 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq

\frq The ‘french’ single guillemets.

```
2136 \ProvideTextCommandDefault{\flq}{%  
2137 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}  
2138 \ProvideTextCommandDefault{\frq}{%  
2139 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq

\frqq The ‘french’ double guillemets.

```
2140 \ProvideTextCommandDefault{\flqq}{%  
2141 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}  
2142 \ProvideTextCommandDefault{\frqq}{%  
2143 \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).

```

2144 \def\umlauthigh{%
2145   \def\bbbl@umlauta##1{\leavevmode\bgroup%
2146     \accent\csname\f@encoding dqpos\endcsname
2147     ##1\bbbl@allowhyphens\egroup}%
2148   \let\bbbl@umlaute\bbbl@umlauta}
2149 \def\umlautlow{%
2150   \def\bbbl@umlauta{\protect\lower@umlaut}}
2151 \def\umlautelowlow{%
2152   \def\bbbl@umlaute{\protect\lower@umlaut}}
2153 \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.

```

2154 \expandafter\ifx\csname U@D\endcsname\relax
2155   \csname newdimen\endcsname U@D
2156 \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.

```

2157 \def\lower@umlaut#1{%
2158   \leavevmode\bgroup
2159     \U@D lex%
2160     {\setbox\z@\hbox{%
2161       \char\csname\f@encoding dqpos\endcsname}%
2162       \dimen@ -.45ex\advance\dimen@ \ht\z@
2163       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2164     \accent\csname\f@encoding dqpos\endcsname
2165     \fontdimen5\font\U@D #1%
2166   \egroup}

```

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

```

2167 \AtBeginDocument{%
2168   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbbl@umlauta{a}}%
2169   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbbl@umlaute{e}}%
2170   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbbl@umlaute{i}}%
2171   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbbl@umlaute{i}}%
2172   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbbl@umlauta{o}}%
2173   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbbl@umlauta{u}}%
2174   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbbl@umlauta{A}}%
2175   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbbl@umlaute{E}}%
2176   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbbl@umlaute{I}}%
2177   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbbl@umlauta{O}}%
2178   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbbl@umlauta{U}}%

```

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

```

2179 \ifx\l@english\undefined
2180   \chardef\l@english\z@
2181 \fi

```

```

2182 % The following is used to cancel rules in ini files (see Amharic).
2183 \ifx\l@unhyphenated\@undefined
2184   \newlanguage\l@unhyphenated
2185 \fi

```

4.16. Layout

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

```

2186 \bbl@trace{Bidi layout}
2187 \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.

```

2188 \bbl@trace{Input engine specific macros}
2189 \ifcase\bbl@engine
2190   \input txtbabel.def
2191 \or
2192   \input luababel.def
2193 \or
2194   \input xebabel.def
2195 \fi
2196 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2197 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2198 \ifx\babelposthyphenation\@undefined
2199   \let\babelposthyphenation\babelprehyphenation
2200   \let\babelpatterns\babelprehyphenation
2201   \let\babelcharproperty\babelprehyphenation
2202 \fi
2203 </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.

```

2204 < *package>
2205 \bbl@trace{Creating languages and reading ini files}
2206 \let\bbl@extend@ini@gobble
2207 \newcommand\babelprovide[2][]{%
2208   \let\bbl@savelangname\languagename
2209   \edef\bbl@savelocaleid{\the\localeid}%
2210   % Set name and locale id
2211   \edef\languagename{#2}%
2212   \bbl@id@assign
2213   % Initialize keys
2214   \bbl@vforeach{captions,date,import,main,script,language,%
2215     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2216     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2217     Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2218     @import}%
2219     {\bbl@csarg\let{KVP@##1}\@nnil}%
2220   \global\let\bbl@release@transforms\@empty
2221   \global\let\bbl@release@casing\@empty
2222   \let\bbl@calendars\@empty
2223   \global\let\bbl@inidata\@empty
2224   \global\let\bbl@extend@ini@gobble
2225   \global\let\bbl@included@inis\@empty
2226   \gdef\bbl@key@list{;}%
2227   \bbl@ifunset\bbl@passto#2}%

```

```

2228     {\def\bb@tempa{#1}}%
2229     {\bb@exp{\def\\bb@tempa{[bb@passto@#2],\unexpanded{#1}}}%
2230 \expandafter\bb@forkv\expandafter{\bb@tempa}{%
2231   \in@{/}{##1}% With /, (re)sets a value in the ini
2232   \ifin@
2233     \bb@renewinikey##1\@{##2}%
2234   \else
2235     \bb@csarg\ifx{KVP@##1}\@nnil\else
2236       \bb@error{unknown-provide-key}{##1}{}}%
2237     \fi
2238     \bb@csarg\def{KVP@##1}{##2}%
2239   \fi}%
2240 \chardef\bb@howloaded=% 0:none; 1:ldf without ini; 2:ini
2241   \bb@ifunset{date#2}\z{\bb@ifunset{bb@llevel@#2}\@ne\tw@}%
2242 % == init ==
2243 \ifx\bb@screset\@undefined
2244   \bb@ldfinit
2245 \fi
2246 % ==
2247 % If there is no import (last wins), use @import (internal, there
2248 % must be just one). To consider any order (because
2249 % \PassOptionsToLocale).
2250 \ifx\bb@KVP@import\@nnil
2251   \let\bb@KVP@import\bb@KVP@import
2252 \fi
2253 % == date (as option) ==
2254 % \ifx\bb@KVP@date\@nnil\else
2255 % \fi
2256 % ==
2257 \let\bb@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2258 \ifcase\bb@howloaded
2259   \let\bb@lbkflag\@empty % new
2260 \else
2261   \ifx\bb@KVP@hyphenrules\@nnil\else
2262     \let\bb@lbkflag\@empty
2263   \fi
2264   \ifx\bb@KVP@import\@nnil\else
2265     \let\bb@lbkflag\@empty
2266   \fi
2267 \fi
2268 % == import, captions ==
2269 \ifx\bb@KVP@import\@nnil\else
2270   \bb@exp{\\bb@ifblank{\bb@KVP@import}}%
2271   {\ifx\bb@initoload\relax
2272     \begingroup
2273       \def\BabelBeforeIni##1##2{\gdef\bb@KVP@import{##1}\endinput}%
2274       \bb@input@texini{##2}%
2275     \endgroup
2276   \else
2277     \xdef\bb@KVP@import{\bb@initoload}%
2278   \fi}%
2279   {}%
2280   \let\bb@KVP@date\@empty
2281 \fi
2282 \let\bb@KVP@captions@\bb@KVP@captions
2283 \ifx\bb@KVP@captions\@nnil
2284   \let\bb@KVP@captions\bb@KVP@import
2285 \fi
2286 % ==
2287 \ifx\bb@KVP@transforms\@nnil\else
2288   \bb@replace\bb@KVP@transforms{ },}%
2289 \fi
2290 % ==

```



```

2291 \ifx\bbk@KVP@mapdot\@nnil\else
2292 \def\bbk@tempa{\@empty}%
2293 \ifx\bbk@KVP@mapdot\bbk@tempa\else
2294 \bbk@exp{\gdef\<bbk@map@. @\language>{\bbk@KVP@mapdot}}}%
2295 \fi
2296 \fi
2297 % Load ini
2298 % -----
2299 \ifcase\bbk@howloaded
2300 \bbk@provide@new{#2}%
2301 \else
2302 \bbk@ifblank{#1}%
2303 {}% With \bbk@load@basic below
2304 {\bbk@provide@renew{#2}}%
2305 \fi
2306 % Post tasks
2307 % -----
2308 % == subsequent calls after the first provide for a locale ==
2309 \ifx\bbk@inidata\@empty\else
2310 \bbk@extend@ini{#2}%
2311 \fi
2312 % == ensure captions ==
2313 \ifx\bbk@KVP@captions\@nnil\else
2314 \bbk@ifunset{bbk@extracaps@#2}%
2315 {\bbk@exp{\bbk@babelensure[exclude=\today]{#2}}}%
2316 {\bbk@exp{\bbk@babelensure[exclude=\today,
2317 include=\bbk@extracaps@#2]{#2}}}%
2318 \bbk@ifunset{bbk@ensure@\language}%
2319 {\bbk@exp{%
2320 \\\DeclareRobustCommand\<bbk@ensure@\language>[1]{%
2321 \\\foreignlanguage{\language}%
2322 {###1}}}%
2323 }%
2324 \bbk@exp{%
2325 \\\bbk@tglobal\<bbk@ensure@\language>%
2326 \\\bbk@tglobal\<bbk@ensure@\language\space>}%
2327 \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.

```

2328 \bbk@load@basic{#2}%
2329 % == script, language ==
2330 % Override the values from ini or defines them
2331 \ifx\bbk@KVP@script\@nnil\else
2332 \bbk@csarg\edef{sname@#2}{\bbk@KVP@script}%
2333 \fi
2334 \ifx\bbk@KVP@language\@nnil\else
2335 \bbk@csarg\edef{lname@#2}{\bbk@KVP@language}%
2336 \fi
2337 \ifcase\bbk@engine\or
2338 \bbk@ifunset{bbk@chrng@\language}{}%
2339 {\directlua{
2340 Babel.set_chranges_b('\bbk@cl{sbc}', '\bbk@cl{chrng}') }}%
2341 \fi
2342 % == Line breaking: intraspace, intrapenalty ==
2343 % For CJK, East Asian, Southeast Asian, if interspace in ini
2344 \ifx\bbk@KVP@intraspace\@nnil\else % We can override the ini or set
2345 \bbk@csarg\edef{intsp@#2}{\bbk@KVP@intraspace}%
2346 \fi
2347 \bbk@provide@intraspace
2348 % == Line breaking: justification ==
2349 \ifx\bbk@KVP@justification\@nnil\else

```

```

2350 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2351 \fi
2352 \ifx\bbl@KVP@linebreaking\@nnil\else
2353 \bbl@xin@{,\bbl@KVP@linebreaking,}%
2354 {,elongated,kashida,cjk,padding,unhyphenated,}%
2355 \ifin@
2356 \bbl@csarg\xdef
2357 {\lbrk@{\language}\lbrk@{\expandafter\@car\bbl@KVP@linebreaking\@nnil}%
2358 \fi
2359 \fi
2360 \bbl@xin@{/e}{/\bbl@cl{\lbrk}}}%
2361 \ifin@{\else\bbl@xin@{/k}{/\bbl@cl{\lbrk}}}\fi
2362 \ifin@\bbl@arabicjust\fi
2363 \bbl@xin@{/p}{/\bbl@cl{\lbrk}}}%
2364 \ifin@AtBeginDocument{\@nameuse{\bbl@tibetanjust}}\fi
2365 % == Line breaking: hyphenate.other.(locale|script) ==
2366 \ifx\bbl@lbrkflag\@empty
2367 \bbl@ifunset{\bbl@hyotl@{\language}}{ }%
2368 {\bbl@csarg\bbl@replace{\hyotl@{\language}}{ }{,}%
2369 \bbl@startcommands*{\language}}{ }%
2370 \bbl@csarg\bbl@foreach{\hyotl@{\language}}{ }%
2371 \ifcase\bbl@engine
2372 \ifnum##1<257
2373 \SetHyphenMap{\BabelLower{##1}{##1}}%
2374 \fi
2375 \else
2376 \SetHyphenMap{\BabelLower{##1}{##1}}%
2377 \fi}%
2378 \bbl@endcommands}%
2379 \bbl@ifunset{\bbl@hyots@{\language}}{ }%
2380 {\bbl@csarg\bbl@replace{\hyots@{\language}}{ }{,}%
2381 \bbl@csarg\bbl@foreach{\hyots@{\language}}{ }%
2382 \ifcase\bbl@engine
2383 \ifnum##1<257
2384 \global\lccode##1=##1\relax
2385 \fi
2386 \else
2387 \global\lccode##1=##1\relax
2388 \fi}}%
2389 \fi
2390 % == Counters: maparabic ==
2391 % Native digits, if provided in ini (TeX level, xe and lua)
2392 \ifcase\bbl@engine\else
2393 \bbl@ifunset{\bbl@dgnat@{\language}}{ }%
2394 {\expandafter\ifx\csname\bbl@dgnat@{\language}\endcsname\@empty\else
2395 \expandafter\expandafter\expandafter
2396 \bbl@setdigits\csname\bbl@dgnat@{\language}\endcsname
2397 \ifx\bbl@KVP@maparabic\@nnil\else
2398 \ifx\bbl@latinarabic\@undefined
2399 \expandafter\let\expandafter\@arabic
2400 \csname\bbl@counter@{\language}\endcsname
2401 \else % i.e., if layout=counters, which redefines \@arabic
2402 \expandafter\let\expandafter\bbl@latinarabic
2403 \csname\bbl@counter@{\language}\endcsname
2404 \fi
2405 \fi
2406 \fi}%
2407 \fi
2408 % == Counters: mapdigits ==
2409 % > luababel.def
2410 % == Counters: alph, Alph ==
2411 \ifx\bbl@KVP@alph\@nnil\else
2412 \bbl@exp{%

```

```

2413     \\bbl@add<bbl@preextras@\\language\\>{%
2414     \\bbl@babel@save\\bbl@alph
2415     \\let\\bbl@alph<bbl@cntr@\\bbl@KVP@alph @\\language\\>}}%
2416 \\fi
2417 \\ifx\\bbl@KVP@Alph@\\nnil\\else
2418     \\bbl@exp{%
2419         \\bbl@add<bbl@preextras@\\language\\>{%
2420         \\bbl@babel@save\\bbl@Alph
2421         \\let\\bbl@Alph<bbl@cntr@\\bbl@KVP@Alph @\\language\\>}}%
2422 \\fi
2423 % == Counters: mapdot ==
2424 \\ifx\\bbl@KVP@mapdot@\\nnil\\else
2425     \\bbl@foreach\\bbl@list@the{%
2426         \\bbl@ifunset{the##1}{}%
2427         {{\\bbl@ncarg\\let\\bbl@tempd{the##1}%
2428         \\bbl@carg\\bbl@sreplace{the##1}{.}{\\bbl@map@lbl{.}}}%
2429         \\expandafter\\ifx\\csname the##1\\endcsname\\bbl@tempd\\else
2430         \\bbl@exp{\\gdef<the##1>{\\[the##1]}}}%
2431         \\fi}}}%
2432 \\edef\\bbl@tempb{enumi,enumii,enumiii,enumiv}%
2433 \\bbl@foreach\\bbl@tempb{%
2434     \\bbl@ifunset{label##1}{}%
2435     {{\\bbl@ncarg\\let\\bbl@tempd{label##1}%
2436     \\bbl@carg\\bbl@sreplace{label##1}{.}{\\bbl@map@lbl{.}}}%
2437     \\expandafter\\ifx\\csname label##1\\endcsname\\bbl@tempd\\else
2438     \\bbl@exp{\\gdef<label##1>{\\[label##1]}}}%
2439     \\fi}}}%
2440 \\fi
2441 % == Casing ==
2442 \\bbl@release@casing
2443 \\ifx\\bbl@KVP@casing@\\nnil\\else
2444     \\bbl@csarg\\xdef{casing@\\language\\}%
2445     {\\nameuse{\\bbl@casing@\\language}\\bbl@maybextx\\bbl@KVP@casing}%
2446 \\fi
2447 % == Calendars ==
2448 \\ifx\\bbl@KVP@calendar@\\nnil
2449     \\edef\\bbl@KVP@calendar{\\bbl@cl{calpr}}}%
2450 \\fi
2451 \\def\\bbl@tempe##1 ##2\\@@{% Get first calendar
2452     \\def\\bbl@tempa{##1}}}%
2453     \\bbl@exp{\\bbl@tempe\\bbl@KVP@calendar\\space\\@@}%
2454 \\def\\bbl@tempe##1.##2.##3\\@@{%
2455     \\def\\bbl@tempc{##1}%
2456     \\def\\bbl@tempb{##2}}}%
2457 \\expandafter\\bbl@tempe\\bbl@tempa..\\@@
2458 \\bbl@csarg\\edef{calpr@\\language\\}{%
2459     \\ifx\\bbl@tempc@empty\\else
2460         calendar=\\bbl@tempc
2461     \\fi
2462     \\ifx\\bbl@tempb@empty\\else
2463         ,variant=\\bbl@tempb
2464     \\fi}%
2465 % == engine specific extensions ==
2466 % Defined in XXXbabel.def
2467 \\bbl@provide@extra{#2}%
2468 % == require.babel in ini ==
2469 % To load or reload the babel-*.tex, if require.babel in ini
2470 \\ifx\\bbl@beforestart\\relax\\else % But not in doc aux or body
2471     \\bbl@ifunset{\\bbl@rqtex@\\language\\}{}%
2472     {\\expandafter\\ifx\\csname bbl@rqtex@\\language\\endcsname\\empty\\else
2473         \\let\\BabelBeforeIni@gobbletwo
2474         \\chardef\\atcatcode=\\catcode`\\@
2475         \\catcode`\\@=11\\relax

```

```

2476     \def\CurrentOption{#2}%
2477     \bbl@input@texini{\bbl@cs{rqtex@}\languagename}}%
2478     \catcode`\@=\atcatcode
2479     \let\atcatcode\relax
2480     \global\bbl@csarg\let{rqtex@}\languagename}\relax
2481     \fi}%
2482 \bbl@foreach\bbl@calendars{%
2483   \bbl@ifunset{\bbl@ca##1}{%
2484     \chardef\atcatcode=\catcode`\@
2485     \catcode`\@=11\relax
2486     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2487     \catcode`\@=\atcatcode
2488     \let\atcatcode\relax}%
2489   {}}%
2490 \fi
2491 % == frenchspacing ==
2492 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2493 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2494 \ifin@
2495   \bbl@extras@wrap{\bbl@pre@fs}%
2496   {\bbl@pre@fs}%
2497   {\bbl@post@fs}%
2498 \fi
2499 % == transforms ==
2500 % > luababel.def
2501 \def\CurrentOption{#2}%
2502 \@nameuse{\bbl@icsave@#2}%
2503 % == main ==
2504 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2505   \let\languagename\bbl@savelangname
2506   \chardef\localeid\bbl@savelocaleid\relax
2507 \fi
2508 % == hyphenrules (apply if current) ==
2509 \ifx\bbl@KVP@hyphenrules\@nnil\else
2510   \ifnum\bbl@savelocaleid=\localeid
2511     \language\@nameuse{l\languagename}%
2512   \fi
2513 \fi}

```

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

```

2514 \def\bbl@provide@new#1{%
2515   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2516   \@namedef{extras#1}{}%
2517   \@namedef{noextras#1}{}%
2518   \bbl@startcommands*{#1}{captions}%
2519   \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2520     \def\bbl@tempb##1{% elt for \bbl@captionslist
2521       \ifx##1\@nnil\else
2522         \bbl@exp{%
2523           \\SetString\\##1{%
2524             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2525         \expandafter\bbl@tempb
2526       \fi}%
2527   \expandafter\bbl@tempb\bbl@captionslist\@nnil
2528 \else
2529   \ifx\bbl@initoload\relax
2530     \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2531   \else
2532     \bbl@read@ini{\bbl@initoload}2% % Same
2533   \fi
2534 \fi
2535 \StartBabelCommands*{#1}{date}%

```

```

2536 \ifx\bbbl@KVP@date\@nnil
2537 \bbbl@exp{%
2538 \SetString\today{\bbbl@nocaption{today}{#1today}}}%
2539 \else
2540 \bbbl@savetoday
2541 \bbbl@savedate
2542 \fi
2543 \bbbl@endcommands
2544 \bbbl@load@basic{#1}%
2545 % == hyphenmins == (only if new)
2546 \bbbl@exp{%
2547 \gdef\<#1hyphenmins>{%
2548 {\bbbl@ifunset{bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2549 {\bbbl@ifunset{bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}%
2550 % == hyphenrules (also in renew) ==
2551 \bbbl@provide@hyphens{#1}%
2552 % == main ==
2553 \ifx\bbbl@KVP@main\@nnil\else
2554 \expandafter\main@language\expandafter{#1}%
2555 \fi}
2556 %
2557 \def\bbbl@provide@renew#1{%
2558 \ifx\bbbl@KVP@captions\@nnil\else
2559 \StartBabelCommands*{#1}{captions}%
2560 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2561 \EndBabelCommands
2562 \fi
2563 \ifx\bbbl@KVP@date\@nnil\else
2564 \StartBabelCommands*{#1}{date}%
2565 \bbbl@savetoday
2566 \bbbl@savedate
2567 \EndBabelCommands
2568 \fi
2569 % == hyphenrules (also in new) ==
2570 \ifx\bbbl@lbkflag\@empty
2571 \bbbl@provide@hyphens{#1}%
2572 \fi
2573 % == main ==
2574 \ifx\bbbl@KVP@main\@nnil\else
2575 \expandafter\main@language\expandafter{#1}%
2576 \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.

```

2577 \def\bbbl@load@basic#1{%
2578 \ifcase\bbbl@howloaded\or\or
2579 \ifcase\csname bbl@llevel@language\endcsname
2580 \bbbl@csarg\let{lname@language}\relax
2581 \fi
2582 \fi
2583 \bbbl@ifunset{bbbl@lname@#1}%
2584 {\def\BabelBeforeIni##1##2{%
2585 \begingroup
2586 \let\bbbl@ini@captions@aux\@gobbletwo
2587 \def\bbbl@inidate #####1.####2.####3.####4\relax #####5####6}%
2588 \bbbl@read@ini{##1}1%
2589 \ifx\bbbl@initoload\relax\endinput\fi
2590 \endgroup}%
2591 \begingroup % boxed, to avoid extra spaces:
2592 \ifx\bbbl@initoload\relax
2593 \bbbl@input@texini{#1}%
2594 \else

```

```

2595     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2596     \fi
2597     \endgroup}%
2598     {}%

```

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).

```

2599 \def\bbl@load@info#1{%
2600   \def\BabelBeforeIni##1##2{%
2601     \begingroup
2602     \bbl@read@ini{##1}0%
2603     \endinput           % babel- .tex may contain onlypreamble's
2604     \endgroup}%        boxed, to avoid extra spaces:
2605     {\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.

```

2606 \def\bbl@provide@hyphens#1{%
2607   \@tempcnta\m@ne % a flag
2608   \ifx\bbl@KVP@hyphenrules\@nnil\else
2609     \bbl@replace\bbl@KVP@hyphenrules{ },}%
2610     \bbl@foreach\bbl@KVP@hyphenrules{%
2611       \ifnum\@tempcnta=\m@ne % if not yet found
2612         \bbl@ifsamestring{##1}{+}%
2613         {\bbl@carg\addlanguage{l@##1}}%
2614         {}%
2615         \bbl@ifunset{l@##1}% After a possible +
2616         {}%
2617         {\@tempcnta\@nameuse{l@##1}}%
2618       \fi}%
2619   \ifnum\@tempcnta=\m@ne
2620     \bbl@warning{%
2621       Requested 'hyphenrules' for '\language' not found:\%
2622       \bbl@KVP@hyphenrules.\%
2623       Using the default value. Reported}%
2624   \fi
2625   \fi
2626   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2627     \ifx\bbl@KVP@captions@\@nnil
2628       \bbl@ifunset\bbl@hyphr{#1}{}% use value in ini, if exists
2629       {\bbl@exp{\@bbl@ifblank{\bbl@cs{hyphr#1}}}%
2630        {}%
2631        {\bbl@ifunset{l@\bbl@cl{hyphr}}%
2632         {}%
2633         {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2634     \fi
2635   \fi
2636   \bbl@ifunset{l@#1}%
2637   {\ifnum\@tempcnta=\m@ne
2638     \bbl@carg\adddialect{l@#1}\language
2639   \else
2640     \bbl@carg\adddialect{l@#1}\@tempcnta
2641   \fi}%
2642   {\ifnum\@tempcnta=\m@ne\else
2643     \global\bbl@carg\chardef{l@#1}\@tempcnta
2644   \fi}}

```

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

```

2645 \def\bbl@input@texini#1{%
2646   \bbl@bsphack
2647   \bbl@exp{%

```

```

2648 \catcode`\\%=14 \catcode`\\\=0
2649 \catcode`\\={1 \catcode`\\\}=2
2650 \lowercase{\\InputIfFileExists{babel-#1.tex}{}}}%
2651 \catcode`\\%=the\catcode`\%\relax
2652 \catcode`\\\=the\catcode`\\\relax
2653 \catcode`\\={the\catcode`\{\relax
2654 \catcode`\\\}=the\catcode`\}\relax}%
2655 \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.

```

2656 \def\bbl@iniline#1\bbl@iniline{%
2657 \ifnextchar[\bbl@inisect{\ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2658 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2659 \def\bbl@iniskip#1\@@{% if starts with ;
2660 \def\bbl@inistore#1=#2\@@{% full (default)
2661 \bbl@trim@def\bbl@tempa{#1}%
2662 \bbl@trim\toks@{#2}%
2663 \bbl@ifsamestring{\bbl@tempa}{\include}%
2664 {\bbl@read@subini{\the\toks@}}%
2665 {\bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2666 \ifin@else
2667 \bbl@xin@{,identification/include.}%
2668 {,\bbl@section/\bbl@tempa}%
2669 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2670 \bbl@exp{%
2671 \\g@addto@macro\\bbl@inidata{%
2672 \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2673 \fi}}
2674 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2675 \bbl@trim@def\bbl@tempa{#1}%
2676 \bbl@trim\toks@{#2}%
2677 \bbl@xin@{.identification.}{.\bbl@section.}%
2678 \ifin@
2679 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2680 \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2681 \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.

```

2682 \def\bbl@loop@ini#1{%
2683 \loop
2684 \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2685 \endlinechar\m@ne
2686 \read#1 to \bbl@line
2687 \endlinechar`\^^M
2688 \ifx\bbl@line\empty\else
2689 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2690 \fi
2691 \repeat}

```

```

2692 %
2693 \def\bbl@read@subini#1{%
2694   \ifx\bbl@readsubstream\@undefined
2695     \csname newread\endcsname\bbl@readsubstream
2696   \fi
2697   \openin\bbl@readsubstream=babel-#1.ini
2698   \ifeof\bbl@readsubstream
2699     \bbl@error{no-ini-file}{#1}{}{}%
2700   \else
2701     {\bbl@loop@ini\bbl@readsubstream}%
2702   \fi
2703   \closein\bbl@readsubstream}
2704 %
2705 \ifx\bbl@readstream\@undefined
2706   \csname newread\endcsname\bbl@readstream
2707 \fi
2708 \def\bbl@read@ini#1#2{%
2709   \global\let\bbl@extend@ini@gobble
2710   \openin\bbl@readstream=babel-#1.ini
2711   \ifeof\bbl@readstream
2712     \bbl@error{no-ini-file}{#1}{}{}%
2713   \else
2714     % == Store ini data in \bbl@inidata ==
2715     \catcode`\ =10 \catcode`\ =12
2716     \catcode`\[=12 \catcode`\]=12 \catcode`\&=12 \catcode`\&=12
2717     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2718     \ifnum#2=\m@ne % Just for the info
2719       \edef\language\tag \bbl@metalang}%
2720     \fi
2721     \bbl@info{\ifnum#2=\m@ne Fetching locale name for tag \bbl@metalang
2722       \else Importing
2723         \ifcase#2font and identification \or basic \fi
2724         data for \language
2725       \fi}%
2726     from babel-#1.ini. Reported}%
2727   \ifnum#2<\@ne
2728     \global\let\bbl@inidata\@empty
2729     \let\bbl@inistore\bbl@inistore@min % Remember it's local
2730   \fi
2731   \def\bbl@section{identification}%
2732   \bbl@exp{%
2733     \\bbl@inistore tag.ini=#1\\@@
2734     \\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\@@}%
2735   \bbl@loop@ini\bbl@readstream
2736   % == Process stored data ==
2737   \ifnum#2=\m@ne
2738     \def\bbl@tempa##1 ##2\@{##1}% Get first name
2739     \def\bbl@elt##1##2##3{%
2740       \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2741       {\edef\language{\bbl@tempa##3 \@}%
2742        \bbl@id@assign
2743        \def\bbl@elt###1###2###3{}}}%
2744     {}}%
2745     \bbl@inidata
2746   \fi
2747   \bbl@csarg\xdef\l@ini{\language}{#1}%
2748   \bbl@read@ini@aux
2749   % == 'Export' data ==
2750   \bbl@ini@exports{#2}%
2751   \global\bbl@csarg\let\inidata{\language}\bbl@inidata
2752   \global\let\bbl@inidata\@empty
2753   \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%
2754   \bbl@tglobal\bbl@ini@loaded

```



```

2755 \fi
2756 \closein\bbl@readstream}
2757 \def\bbl@read@ini@aux{%
2758 \let\bbl@savestrings\empty
2759 \let\bbl@savetoday\empty
2760 \let\bbl@savestate\empty
2761 \def\bbl@elt##1##2##3{%
2762 \def\bbl@section{##1}%
2763 \in@{=date.}{=##1}% Find a better place
2764 \ifin@
2765 \bbl@ifunset{bbl@inikv@##1}%
2766 {\bbl@ini@calendar{##1}}%
2767 {}%
2768 \fi
2769 \bbl@ifunset{bbl@inikv@##1}{}%
2770 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2771 \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.

```

2772 \def\bbl@extend@ini@aux#1{%
2773 \bbl@startcommands*{#1}{captions}%
2774 % Activate captions/... and modify exports
2775 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2776 \setlocalecaption{#1}{##1}{##2}}%
2777 \def\bbl@inikv@captions##1##2{%
2778 \bbl@ini@captions@aux{##1}{##2}}%
2779 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2780 \def\bbl@exportkey##1##2##3{%
2781 \bbl@ifunset{bbl@kv@##2}{}%
2782 {\expandafter\ifx\csname bbl@kv@##2\endcsname\empty\else
2783 \bbl@exp{\global\let\<bbl@##1@languagename>\<bbl@kv@##2>}}%
2784 \fi}}%
2785 % As with \bbl@read@ini, but with some changes
2786 \bbl@read@ini@aux
2787 \bbl@ini@exports\tw@
2788 % Update inidata@lang by pretending the ini is read.
2789 \def\bbl@elt##1##2##3{%
2790 \def\bbl@section{##1}%
2791 \bbl@iniline##2=##3\bbl@iniline}%
2792 \csname bbl@inidata@#1\endcsname
2793 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2794 \StartBabelCommands*{#1}{date}% And from the import stuff
2795 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2796 \bbl@savetoday
2797 \bbl@savestate
2798 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2799 \def\bbl@ini@calendar#1{%
2800 \lowercase{\def\bbl@tempa{=##1=}}%
2801 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2802 \bbl@replace\bbl@tempa{=date.}{}%
2803 \in@{.licr=}{#1=}%
2804 \ifin@
2805 \ifcase\bbl@engine
2806 \bbl@replace\bbl@tempa{.licr=}{}%
2807 \else
2808 \let\bbl@tempa\relax
2809 \fi
2810 \fi
2811 \ifx\bbl@tempa\relax\else
2812 \bbl@replace\bbl@tempa{=}{}%
2813 \ifx\bbl@tempa\empty\else

```

```

2814 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2815 \fi
2816 \bbl@exp{%
2817 \def<\bbl@inikv@#1>####1####2{%
2818 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2819 \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).

```

2820 \def\bbl@renewinikey#1/#2\@#3{%
2821 \global\let\bbl@extend@ini\bbl@extend@ini@aux
2822 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2823 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2824 \bbl@trim\toks@{#3}% value
2825 \bbl@exp{%
2826 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2827 \\\gaddto@macro\\bbl@inidata{%
2828 \\\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.

```

2829 \def\bbl@exportkey#1#2#3{%
2830 \bbl@ifunset{\bbl@kv@#2}%
2831 {\bbl@csarg\gdef{#1@\language name}{#3}}%
2832 {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2833 \bbl@csarg\gdef{#1@\language name}{#3}%
2834 \else
2835 \bbl@exp{\global\let<\bbl@#1@\language name><\bbl@kv@#2>}%
2836 \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.

```

2837 \def\bbl@iniwarning#1{%
2838 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2839 {\bbl@warning{%
2840 From babel-\bbl@cs{lini@\language name}.ini:\\%
2841 \bbl@cs{@kv@identification.warning#1}\\%
2842 Reported}}}
2843 %
2844 \let\bbl@release@transforms\@empty
2845 \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).

```

2846 \def\bbl@ini@exports#1{%
2847 % Identification always exported
2848 \bbl@iniwarning{}%
2849 \ifcase\bbl@engine
2850 \bbl@iniwarning{.pdfLaTeX}%
2851 \or

```

```

2852 \bbl@iniwarning{.lualatex}%
2853 \or
2854 \bbl@iniwarning{.xelatex}%
2855 \fi%
2856 \bbl@exportkey{lllevel}{identification.load.level}{}%
2857 \bbl@exportkey{elname}{identification.name.english}{}%
2858 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2859 {\csname bbl@elname@language\endcsname}}%
2860 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2861 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2862 \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2863 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2864 \bbl@exportkey{esname}{identification.script.name}{}%
2865 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2866 {\csname bbl@esname@language\endcsname}}%
2867 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2868 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2869 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2870 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2871 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2872 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2873 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2874 % Also maps bcp47 -> language
2875 \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
2876 \ifcase\bbl@engine\or
2877 \directlua{%
2878 Babel.locale_props[\the\bbl@cs{id@language}].script
2879 = '\bbl@cl{sbc}}}%
2880 \fi
2881 % Conditional
2882 \ifnum#1>\z@ % -1 or 0 = only info, 1 = basic, 2 = (re)new
2883 \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2884 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2885 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2886 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2887 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2888 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2889 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2890 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2891 \bbl@exportkey{intsp}{typography.intraspace}{}%
2892 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2893 \bbl@exportkey{chrng}{characters.ranges}{}%
2894 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2895 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2896 \ifnum#1=\tw@ % only (re)new
2897 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2898 \bbl@tglobal\bbl@savetoday
2899 \bbl@tglobal\bbl@savestate
2900 \bbl@savestrings
2901 \fi
2902 \fi}

```

4.20. Processing keys in ini

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

```

2903 \def\bbl@inikv#1#2{%      key=value
2904 \toks@{#2}%              This hides #'s from ini values
2905 \bbl@csarg\edef{kv@bbl@section.#1}{\the\toks@}}

```

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

```

2906 \let\bbl@inikv@identification\bbl@inikv
2907 \let\bbl@inikv@date\bbl@inikv
2908 \let\bbl@inikv@typography\bbl@inikv

```

2909 \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.

```
2910 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\@empty x-\fi}
2911 \def\bbl@inikv@characters#1#2{%
2912   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2913   {\bbl@exp{%
2914     \\g@addto@macro\\bbl@release@casing{%
2915       \\bbl@casemapping}{\language}\unexpanded{#2}}}%
2916   {\in{$casing.}{#1}% e.g., casing.Uv = uV
2917   \ifin@
2918     \lowercase{\def\bbl@tempb{#1}%
2919     \bbl@replace\bbl@tempb{casing.}{}%
2920     \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2921       \\bbl@casemapping
2922       {\\bbl@maybextx\bbl@tempb}\language}\unexpanded{#2}}}%
2923   \else
2924     \bbl@inikv{#1}{#2}%
2925   \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’.

```
2926 \def\bbl@inikv@counters#1#2{%
2927   \bbl@ifsamestring{#1}{digits}%
2928   {\bbl@error{digits-is-reserved}{}}}%
2929   {%
2930   \def\bbl@tempc{#1}%
2931   \bbl@trim@def{\bbl@tempb*}{#2}%
2932   \in@{.1$}{#1$}%
2933   \ifin@
2934     \bbl@replace\bbl@tempc{.1}{}%
2935     \bbl@csarg\protected@xdef{cnt@bbl@tempc @\language}{%
2936       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2937   \fi
2938   \in@{.F.}{#1}%
2939   \ifin@else\in@{.S.}{#1}\fi
2940   \ifin@
2941     \bbl@csarg\protected@xdef{cnt@#1@\language}\bbl@tempb*}%
2942   \else
2943     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2944     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
2945     \bbl@csarg{\global\expandafter\let}{cnt@#1@\language}\bbl@tempa
2946   \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.

```
2947 \ifcase\bbl@engine
2948   \bbl@csarg\def{inikv@captions.licr}#1#2{%
2949     \bbl@ini@captions@aux{#1}{#2}}
2950 \else
2951   \def\bbl@inikv@captions#1#2{%
2952     \bbl@ini@captions@aux{#1}{#2}}
2953 \fi
```

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

```
2954 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2955   \bbl@replace\bbl@tempa{.template}{}%
2956   \def\bbl@toreplace{#1}{}%
2957   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
2958   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
```

```

2959 \bbl@replace\bbl@toreplace{[]}{\csname the}%
2960 \bbl@replace\bbl@toreplace{[]]}{name\endcsname{}}}%
2961 \bbl@replace\bbl@toreplace{[]]}{\endcsname{}}}%
2962 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2963 \ifin@
2964   \@nameuse{bbl@patch\bbl@tempa}%
2965   \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2966 \fi
2967 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2968 \ifin@
2969   \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2970   \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2971     \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\\language}%
2972     {[fnum@\bbl@tempa]}%
2973     {\\\@nameuse{bbl@\bbl@tempa fmt@\\\language}}}}}%
2974 \fi}
2975 %
2976 \def\bbl@ini@captions@aux#1#2{%
2977   \bbl@trim@def\bbl@tempa{#1}%
2978   \bbl@xin@{.template}{\bbl@tempa}%
2979   \ifin@
2980     \bbl@ini@captions@template{#2}\language
2981   \else
2982     \bbl@ifblank{#2}%
2983     {\bbl@exp{%
2984       \toks@{\\bbl@nocaption{\bbl@tempa name}{\language\bbl@tempa name}}}%
2985       {\bbl@trim\toks@{#2}}}%
2986     \bbl@exp{%
2987       \\\bbl@add\\bbl@savestrings{%
2988         \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
2989       \toks@\expandafter\bbl@captionslist}%
2990     \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}%
2991     \ifin@else
2992       \bbl@exp{%
2993         \\\bbl@add\<bbl@extracaps@\language>{\<\bbl@tempa name>}%
2994         \\\bbl@tglobal\<bbl@extracaps@\language>}%
2995       \fi
2996     \fi}

```

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

```

2997 \def\bbl@list@the{%
2998   part,chapter,section,subsection,subsubsection,paragraph,%
2999   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3000   table,page,footnote,mpfootnote,mpfn}
3001 %
3002 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3003   \bbl@ifunset{bbl@map@#1@\language}%
3004   {\@nameuse{#1}}%
3005   {\@nameuse{bbl@map@#1@\language}}}
3006 %
3007 \def\bbl@map@lbl#1{% #1:a sign, eg, .
3008   \ifin@csname#1\else
3009     \bbl@ifunset{bbl@map@@#1@\language}%
3010     {#1}%
3011     {\@nameuse{bbl@map@@#1@\language}}}%
3012 \fi}
3013 %
3014 \def\bbl@inikv@labels#1#2{%
3015   \in@{.map}{#1}%
3016   \ifin@
3017     \in@{,dot.map,}{, #1,}%
3018   \ifin@
3019     \global\@namedef{bbl@map@@.@\language}{#2}%

```

```

3020 \fi
3021 \ifx\bbk@KVP@labels\@nnil\else
3022 \bbk@xin@{ map }{ \bbk@KVP@labels\space}%
3023 \ifin@
3024 \def\bbk@tempc{#1}%
3025 \bbk@replace\bbk@tempc{.map}{}%
3026 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3027 \bbk@exp{%
3028 \gdef\<bbk@map@\bbk@tempc @\language\name>%
3029 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3030 \bbk@foreach\bbk@list@the{%
3031 \bbk@ifunset{the##1}{}%
3032 {\bbk@ncarg\let\bbk@tempd{the##1}%
3033 \bbk@exp{%
3034 \\bbk@sreplace\<the##1>%
3035 {\<\bbk@tempc>{##1}}%
3036 {\bbk@map@cnt{\bbk@tempc}{##1}}%
3037 \\bbk@sreplace\<the##1>%
3038 {\<@empty @\bbk@tempc>\<c@##1>%
3039 {\bbk@map@cnt{\bbk@tempc}{##1}}%
3040 \\bbk@sreplace\<the##1>%
3041 {\csname @\bbk@tempc\endcsname\<c@##1>%
3042 {\bbk@map@cnt{\bbk@tempc}{##1}}}}%
3043 \expandafter\ifx\csname the##1\endcsname\bbk@tempd\else
3044 \bbk@exp{\gdef\<the##1>{\[the##1]}}%
3045 \fi}}%
3046 \fi
3047 \fi
3048 %
3049 \else
3050 % The following code is still under study. You can test it and make
3051 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3052 % language dependent.
3053 \in@{enumerate.}{#1}%
3054 \ifin@
3055 \def\bbk@tempa{#1}%
3056 \bbk@replace\bbk@tempa{enumerate.}{}%
3057 \def\bbk@toreplace{#2}%
3058 \bbk@replace\bbk@toreplace{[ ]}{\nobreakspace{}}%
3059 \bbk@replace\bbk@toreplace{[ ]}{\csname the}%
3060 \bbk@replace\bbk@toreplace{[ ]}{\endcsname{}}%
3061 \toks@{\expandafter\bbk@toreplace}%
3062 \bbk@exp{%
3063 \\bbk@add\<extras\language\name>%
3064 \\babel@save\<labelenum\romannumeral\bbk@tempa>%
3065 \def\<labelenum\romannumeral\bbk@tempa>{\the\toks@}%
3066 \\bbk@tglobal\<extras\language\name>%
3067 \fi
3068 \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.

```

3069 \def\bbk@chapttype{chapter}
3070 \ifx\@makechapterhead\@undefined
3071 \let\bbk@patchchapter\relax
3072 \else\ifx\thechapter\@undefined
3073 \let\bbk@patchchapter\relax
3074 \else\ifx\ps@headings\@undefined
3075 \let\bbk@patchchapter\relax
3076 \else
3077 \def\bbk@patchchapter{%

```

```

3078 \global\let\bbl@patchchapter\relax
3079 \gdef\bbl@chfmt{%
3080   \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3081   {\@chapapp\space\thechapter}%
3082   {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}%
3083 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3084 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3085 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3086 \bbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3087 \bbl@tglobal\appendix
3088 \bbl@tglobal\ps@headings
3089 \bbl@tglobal\chaptermark
3090 \bbl@tglobal\makechapterhead}
3091 \let\bbl@patchappendix\bbl@patchchapter
3092 \fi\fi\fi
3093 \ifx\@part\undefined
3094   \let\bbl@patchpart\relax
3095 \else
3096   \def\bbl@patchpart{%
3097     \global\let\bbl@patchpart\relax
3098     \gdef\bbl@partformat{%
3099       \bbl@ifunset{\bbl@partfmt@\language}%
3100       {\partname\nobreakspace\thepart}%
3101       {\@nameuse{\bbl@partfmt@\language}}}%
3102     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3103     \bbl@tglobal\@part}
3104 \fi

```

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

```

3105 \let\bbl@calendar\@empty
3106 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3107 \def\bbl@localedate#1#2#3#4{%
3108   \begingroup
3109     \edef\bbl@they{#2}%
3110     \edef\bbl@them{#3}%
3111     \edef\bbl@thed{#4}%
3112     \edef\bbl@tempe{%
3113       \bbl@ifunset{\bbl@calpr@\language}{\bbl@cl{calpr}},%
3114       #1}%
3115     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3116     \bbl@replace\bbl@tempe{ }{}%
3117     \bbl@replace\bbl@tempe{convert}{convert=}%
3118     \let\bbl@ld@calendar\@empty
3119     \let\bbl@ld@variant\@empty
3120     \let\bbl@ld@convert\relax
3121     \def\bbl@tempb##1=##2\@@{\@namedef{\bbl@ld@##1}{##2}}%
3122     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3123     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3124     \ifx\bbl@ld@calendar\@empty\else
3125       \ifx\bbl@ld@convert\relax\else
3126         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3127         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3128       \fi
3129     \fi
3130     \@nameuse{\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3131     \edef\bbl@calendar{% Used in \month..., too
3132       \bbl@ld@calendar
3133       \ifx\bbl@ld@variant\@empty\else
3134         .\bbl@ld@variant
3135       \fi}%
3136     \bbl@cased
3137     {\@nameuse{\bbl@date@\language @\bbl@calendar}}%

```

```

3138         \bbl@they\bbl@them\bbl@thed}%
3139 \endgroup}
3140 %
3141 \def\bbl@printdate#1{%
3142   \ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}}
3143 \def\bbl@printdate@i#1[#2]#3#4#5{%
3144   \bbl@usedategroupttrue
3145   \@nameuse{bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}}%
3146 %
3147 e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3148 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3149   \bbl@trim@def\bbl@tempa{#1.#2}%
3150   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3151   {\bbl@trim@def\bbl@tempa{#3}%
3152     \bbl@trim\toks@{#5}%
3153     \@temptokena\expandafter{\bbl@savestate}%
3154     \bbl@exp{% Reverse order - in ini last wins
3155       \def\\bbl@savestate{%
3156         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3157         \the\@temptokena}}}%
3158   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3159     {\lowercase{\def\bbl@tempb{#6}}}%
3160     \bbl@trim@def\bbl@toreplace{#5}%
3161     \bbl@TG@@date
3162     \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3163     \ifx\bbl@savestate\empty
3164       \bbl@exp{%
3165         \\AfterBabelCommands{%
3166           \gdef\<\language name date>{\\protect\<\language name date >}%
3167           \gdef\<\language name date >{\\bbl@printdate{\language name}}}%
3168         \def\\bbl@savestate{%
3169           \\SetString\\today{%
3170             \<\language name date>[convert]%
3171             {\\the\year}{\\the\month}{\\the\day}}}}}%
3172     \fi}%
3173   {}}}}

```

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).

```

3174 \let\bbl@calendar\empty
3175 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3176   \@nameuse{bbl@ca@#2}#1@@}
3177 \newcommand\babelDateSpace{\nobreakspace}
3178 \newcommand\babelDateDot{.\@}
3179 \newcommand\babelDated[1][{\number#1}]
3180 \newcommand\babelDatedd[1][{\ifnum#1<10 0\fi\number#1}]
3181 \newcommand\babelDateM[1][{\number#1}]
3182 \newcommand\babelDateMM[1][{\ifnum#1<10 0\fi\number#1}]
3183 \newcommand\babelDateMMM[1][{\number#1}]
3184 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3185 \newcommand\babelDatey[1][{\number#1}]
3186 \newcommand\babelDateyy[1][{\number#1}]
3187 \ifnum#1<10 0\number#1 %
3188 \else\ifnum#1<100 \number#1 %
3189 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3190 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3191 \else
3192   \bbl@error{limit-two-digits}{\number#1}%
3193   \fi\fi\fi\fi}
3194 \newcommand\babelDateyyyy[1][{\number#1}]

```



```

3195 \newcommand\BabelDateU[1]{\number#1}%
3196 \def\bb@l@replace@finish@iii#1{%
3197   \bb@exp{\def\#1###1###2###3{\the\toks@}}
3198 \def\bb@l@TG@date{%
3199   \bb@replace\bb@toreplace{[ ]}{\BabelDateSpace{}}%
3200   \bb@replace\bb@toreplace{.}{\BabelDateDot{}}%
3201   \bb@replace\bb@toreplace{[d]}{\BabelDated{###3}}%
3202   \bb@replace\bb@toreplace{[dd]}{\BabelDateddd{###3}}%
3203   \bb@replace\bb@toreplace{[M]}{\BabelDateM{###2}}%
3204   \bb@replace\bb@toreplace{[MM]}{\BabelDateMM{###2}}%
3205   \bb@replace\bb@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3206   \bb@replace\bb@toreplace{[y]}{\BabelDatey{###1}}%
3207   \bb@replace\bb@toreplace{[yy]}{\BabelDateyy{###1}}%
3208   \bb@replace\bb@toreplace{[yyyy]}{\BabelDateyyy{###1}}%
3209   \bb@replace\bb@toreplace{[U]}{\BabelDateU{###1}}%
3210   \bb@replace\bb@toreplace{[y|]}{\bb@datecctr[###1|]}%
3211   \bb@replace\bb@toreplace{[U|]}{\bb@datecctr[###1|]}%
3212   \bb@replace\bb@toreplace{[m|]}{\bb@datecctr[###2|]}%
3213   \bb@replace\bb@toreplace{[d|]}{\bb@datecctr[###3|]}%
3214   \bb@replace@finish@iii\bb@toreplace}
3215 \def\bb@l@datecctr{\expandafter\bb@xdatecctr\expandafter}
3216 \def\bb@l@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.

```

3217 \AddToHook{begindocument/before}{%
3218   \let\bb@l@normalsf\normalsfcodes
3219   \let\normalsfcodes\relax}
3220 \AtBeginDocument{%
3221   \ifx\bb@l@normalsf\@empty
3222     \ifnum\sfcodes\@m
3223       \let\normalsfcodes\frenchspacing
3224     \else
3225       \let\normalsfcodes\nonfrenchspacing
3226     \fi
3227   \else
3228     \let\normalsfcodes\bb@l@normalsf
3229   \fi}

```

Transforms.

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

```

3230 \bb@l@csarg\let{inikv@transforms.prehyphenation}\bb@inikv
3231 \bb@l@csarg\let{inikv@transforms.posthyphenation}\bb@inikv
3232 \def\bb@l@transforms@aux#1#2#3#4,#5\relax{%
3233   #1[#2]{#3}{#4}{#5}}
3234 \begingroup
3235   \catcode`\%=12
3236   \catcode`\&=14
3237   \gdef\bb@l@transforms#1#2#3{%&
3238     \directlua{
3239       local str = [==[#2]==]
3240       str = str:gsub('%.%d+%.%d+$', '')
3241       token.set_macro('babeltempa', str)
3242     }&
3243     \def\babeltempc{ }&
3244     \bb@l@xin{,\babeltempa,}{,\bb@l@KVP@transforms,}&
3245     \ifin@else

```

```

3246     \bbl@xin@{: \babeltempa,}{, \bbl@KVP@transforms,}&%
3247 \fi
3248 \ifin@
3249     \bbl@foreach\bbl@KVP@transforms{&%
3250         \bbl@xin@{: \babeltempa,}{, ##1,}&%
3251         \ifin@ &% font:font:transform syntax
3252             \directlua{
3253                 local t = {}
3254                 for m in string.gmatch('##1'..' ':'(.)') do
3255                     table.insert(t, m)
3256                 end
3257                 table.remove(t)
3258                 token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3259             }&%
3260         \fi}&%
3261 \in@{.0$}{#2$}&%
3262 \ifin@
3263     \directlua{&% (\attribute) syntax
3264         local str = string.match([[ \bbl@KVP@transforms]],
3265             '%([^(%[-])%)[^)]-\babeltempa')
3266         if str == nil then
3267             token.set_macro('babeltempb', '')
3268         else
3269             token.set_macro('babeltempb', ', attribute=' .. str)
3270         end
3271     }&%
3272 \toks@{#3}&%
3273 \bbl@exp{&%
3274     \\g@addto@macro\\ \bbl@release@transforms{&%
3275         \relax &% Closes previous \bbl@transforms@aux
3276         \\ \bbl@transforms@aux
3277         \\ #1{label=\babeltempa\babeltempb\babeltempc}&%
3278         {\language\the\toks@}}&%
3279 \else
3280     \g@addto@macro\bbl@release@transforms{, {#3}}&%
3281 \fi
3282 \fi}
3283 \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.

```

3284 \def\bbl@provide@lsys#1{%
3285     \bbl@ifunset{bbl@lname@#1}%
3286     {\bbl@load@info{#1}}%
3287     }%
3288 \bbl@csarg\let{lsys@#1}\@empty
3289 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{%
3290 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3291 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3292 \bbl@ifunset{bbl@lname@#1}}%
3293 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3294 \ifcase\bbl@engine\or\or
3295     \bbl@ifunset{bbl@prehc@#1}{%
3296         {\bbl@exp{\\ \bbl@ifblank{\bbl@cs{prehc@#1}}}%
3297         }%
3298         {\ifx\bbl@xenoxyph\undefined
3299             \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3300             \ifx\AtBeginDocument\@notprerr
3301                 \expandafter\@secondoftwo % to execute right now

```

```

3302         \fi
3303         \AtBeginDocument{%
3304             \bbl@patchfont{\bbl@xenohyph}%
3305             {\expandafter\select@language\expandafter{\language\language}}}%
3306         \fi}}%
3307 \fi
3308 \bbl@csarg\bbl@tglobal{\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 \TeX . Non-digits characters are kept. The first macro is the generic “localized” command.

[illegible]

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

```

3340 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3341   \ifx\\#1%
3342     \bbl@exp{%
3343       \def\\bbl@tempa####1{%
3344         \<ifcase>####1\space\the\toks@\<else>\\<ctrerr>\<fi>}}%
3345   \else
3346     \toks@\expandafter{\the\toks@\or #1}%
3347     \expandafter\bbl@buildifcase
3348   \fi}

```

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

```
3349 \newcommand\localnumeral[2]{%
3350   \bbl@ifunset{bbl@cntr@#1@}\languagename}%
```

```

3351     {#2}%
3352     {\bbl@cs{cntr@#1@\language}\{#2}}
3353 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3354 \newcommand\localecounter[2]{%
3355   \expandafter\bbl@localecntr
3356   \expandafter{\number\csname c@#2\endcsname}\{#1}}
3357 \def\bbl@alphnumeral#1#2{%
3358   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3359 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3360   \ifcase\@car#8\@nil\or    % Currently <10000, but prepared for bigger
3361     \bbl@alphnumeral@ii{#9}00000#1\or
3362     \bbl@alphnumeral@ii{#9}00000#1#2\or
3363     \bbl@alphnumeral@ii{#9}00000#1#2#3\or
3364     \bbl@alphnumeral@ii{#9}00000#1#2#3#4\else
3365     \bbl@alphnum@invalid{>9999}%
3366   \fi}
3367 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3368   \bbl@ifunset{\bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3369   {\bbl@cs{cntr@#1.4@\language}#5%
3370    \bbl@cs{cntr@#1.3@\language}#6%
3371    \bbl@cs{cntr@#1.2@\language}#7%
3372    \bbl@cs{cntr@#1.1@\language}#8%
3373    \ifnum#6#7#8>\z@
3374      \bbl@ifunset{\bbl@cntr@#1.S.321@\language}{}%
3375      {\bbl@cs{cntr@#1.S.321@\language}}%
3376    \fi}%
3377   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}}
3378 \def\bbl@alphnum@invalid#1{%
3379   \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```

3380 \newcommand\BabelUppercaseMapping[3]{%
3381   \DeclareUppercaseMapping[\@nameuse{\bbl@casing@#1}]{#2}{#3}}
3382 \newcommand\BabelTitlecaseMapping[3]{%
3383   \DeclareTitlecaseMapping[\@nameuse{\bbl@casing@#1}]{#2}{#3}}
3384 \newcommand\BabelLowercaseMapping[3]{%
3385   \DeclareLowercaseMapping[\@nameuse{\bbl@casing@#1}]{#2}{#3}}

  The parser for casing and casing.⟨variant⟩.
3386 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3387   \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3388 \else
3389   \def\bbl@uftocode#1{\expandafter`\string#1}
3390 \fi
3391 \def\bbl@casemapping#1#2#3{% 1:variant
3392   \def\bbl@tempa##1 ##2{% Loop
3393     \bbl@casemapping@i{##1}%
3394     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3395   \edef\bbl@templ{\@nameuse{\bbl@casing@#2}#1}% Language code
3396   \def\bbl@tempe{0}% Mode (upper/lower...)
3397   \def\bbl@tempc{#3}% Casing list
3398   \expandafter\bbl@tempa\bbl@tempc\@empty}
3399 \def\bbl@casemapping@i#1{%
3400   \def\bbl@tempb{#1}%
3401   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3402     \@nameuse{regex_replace_all:nnN}%
3403     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\0}}\bbl@tempb
3404   \else
3405     \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb
3406   \fi
3407   \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3408 \def\bbl@casemapping@ii#1#2#3\@@{%
3409   \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>

```

```

3410 \ifin@
3411 \edef\bbl@tempe{%
3412 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3413 \else
3414 \ifcase\bbl@tempe\relax
3415 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3416 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3417 \or
3418 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3419 \or
3420 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3421 \or
3422 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3423 \fi
3424 \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.

```

3425 \def\bbl@localeinfo#1#2{%
3426 \bbl@ifunset\bbl@info@#2}{#1}%
3427 {\bbl@ifunset\bbl@csname\bbl@info@#2\endcsname @\language name}{#1}%
3428 {\bbl@cs{csname\bbl@info@#2\endcsname @\language name}}}%
3429 \newcommand\localeinfo[1]{%
3430 \ifx*#1\@empty
3431 \bbl@afterelse\bbl@localeinfo}%
3432 \else
3433 \bbl@localeinfo
3434 {\bbl@error{no-ini-info}}{}{}{}%
3435 {#1}%
3436 \fi}
3437 % \@namedef{\bbl@info@name.locale}{\lcname}
3438 \@namedef{\bbl@info@tag.ini}{\lini}
3439 \@namedef{\bbl@info@name.english}{\elname}
3440 \@namedef{\bbl@info@name.opentype}{\lname}
3441 \@namedef{\bbl@info@tag.bcp47}{\tbcp}
3442 \@namedef{\bbl@info@language.tag.bcp47}{\lbc}
3443 \@namedef{\bbl@info@tag.opentype}{\lotf}
3444 \@namedef{\bbl@info@script.name}{\esname}
3445 \@namedef{\bbl@info@script.name.opentype}{\sname}
3446 \@namedef{\bbl@info@script.tag.bcp47}{\sbcp}
3447 \@namedef{\bbl@info@script.tag.opentype}{\sotf}
3448 \@namedef{\bbl@info@region.tag.bcp47}{\rbcp}
3449 \@namedef{\bbl@info@variant.tag.bcp47}{\vbcp}
3450 \@namedef{\bbl@info@extension.t.tag.bcp47}{\extt}
3451 \@namedef{\bbl@info@extension.u.tag.bcp47}{\extu}
3452 \@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.

```

3453 << *More package options >> ≡
3454 \DeclareOption{ensureinfo=off}{}
3455 << /More package options >>
3456 \let\BabelEnsureInfo\relax

```

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

```

3457 \newcommand\getlocaleproperty{%
3458 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3459 \def\bbl@getproperty@s#1#2#3{%
3460 \let#1\relax
3461 \def\bbl@elt##1##2##3{%
3462 \bbl@ifsamestring{##1/##2}{#3}%
3463 {\providecommand#1{##3}%

```

```

3464 \def\bbl@elt####1####2####3{}}%
3465 {}}%
3466 \bbl@cs{inidata@#2}}%
3467 \def\bbl@getproperty@x#1#2#3{%
3468 \bbl@getproperty@s{#1}{#2}{#3}%
3469 \ifx#1\relax
3470 \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3471 \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`.

```

3472 \let\bbl@ini@loaded\@empty
3473 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3474 \def\ShowLocaleProperties#1{%
3475 \typeout{}}%
3476 \typeout{*** Properties for language '#1' ***}
3477 \def\bbl@elt####1####2####3{\typeout{##1/##2 = \unexpanded{##3}}}%
3478 \@nameuse{\bbl@inidata@#1}%
3479 \typeout{*****}}

```

4.26. BCP 47 related commands

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

```

3480 \newif\ifbbl@bcppallowed
3481 \bbl@bcppallowedfalse
3482 \def\bbl@autoload@options{@import}
3483 \def\bbl@provide@locale{%
3484 \ifx\babelprovide\@undefined
3485 \bbl@error{base-on-the-fly}{}}}%
3486 \fi
3487 \let\bbl@auxname\language
3488 \ifbbl@bcptoname
3489 \bbl@ifunset{\bbl@bcp@map@\language}{}}% Move uplevel??
3490 {\edef\language{\@nameuse{\bbl@bcp@map@\language}}}%
3491 \let\localename\language}%
3492 \fi
3493 \ifbbl@bcppallowed
3494 \expandafter\ifx\csname date\language\endcsname\relax
3495 \expandafter
3496 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
3497 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3498 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3499 \let\localename\language
3500 \expandafter\ifx\csname date\language\endcsname\relax
3501 \let\bbl@initoload\bbl@bcp
3502 \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3503 \let\bbl@initoload\relax
3504 \fi
3505 \bbl@csarg\xdef{\bcp@map@\bbl@bcp}{\localename}%
3506 \fi
3507 \fi
3508 \fi
3509 \expandafter\ifx\csname date\language\endcsname\relax
3510 \IfFileExists{babel-\language.tex}%
3511 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3512 {}%
3513 \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.`\langle s \rangle` 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`.

```

3514 \providecommand\BCPdata{}
3515 \ifx\renewcommand\undefined\else
3516   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3517   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3518     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3519     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3520     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language\language}%
3521   \def\bbl@bcpdata@ii#1#2{%
3522     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3523     {\bbl@error{unknown-ini-field}{#1}{}}}%
3524     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3525     {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3526 \fi
3527 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3528 \@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.

```

3529 \newcommand\babeladjust[1]{%
3530   \bbl@forkv{#1}{%
3531     \bbl@ifunset{bbl@ADJ@##1@##2}%
3532     {\bbl@cs{ADJ@##1}{##2}}%
3533     {\bbl@cs{ADJ@##1@##2}}}
3534 %
3535 \def\bbl@adjust@lua#1#2{%
3536   \ifvmode
3537     \ifnum\currentgrouplevel=\z@
3538       \directlua{ Babel.#2 }%
3539       \expandafter\expandafter\expandafter@gobble
3540     \fi
3541   \fi
3542   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3543 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3544   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3545 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3546   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3547 \@namedef{bbl@ADJ@bidi.text@on}{%
3548   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3549 \@namedef{bbl@ADJ@bidi.text@off}{%
3550   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3551 \@namedef{bbl@ADJ@bidi.math@on}{%
3552   \let\bbl@noamsmath\@empty}
3553 \@namedef{bbl@ADJ@bidi.math@off}{%
3554   \let\bbl@noamsmath\relax}
3555 %
3556 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3557   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3558 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3559   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3560 %
3561 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3562   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3563 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3564   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3565 \@namedef{bbl@ADJ@linebreak.cjk@on}{%

```

```

3566 \bbl@adjust@lua{linebreak}{CJK_enabled=true}}
3567 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3568 \bbl@adjust@lua{linebreak}{CJK_enabled=false}}
3569 \@namedef{bbl@ADJ@justify.arabic@on}{%
3570 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3571 \@namedef{bbl@ADJ@justify.arabic@off}{%
3572 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3573 %
3574 \def\bbl@adjust@layout#1{%
3575 \ifvmode
3576 #1%
3577 \expandafter\@gobble
3578 \fi
3579 {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3580 \@namedef{bbl@ADJ@layout.tabular@on}{%
3581 \ifnum\bbl@tabular@mode=\tw@
3582 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3583 \else
3584 \chardef\bbl@tabular@mode\@ne
3585 \fi}
3586 \@namedef{bbl@ADJ@layout.tabular@off}{%
3587 \ifnum\bbl@tabular@mode=\tw@
3588 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3589 \else
3590 \chardef\bbl@tabular@mode\@z@
3591 \fi}
3592 \@namedef{bbl@ADJ@layout.lists@on}{%
3593 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3594 \@namedef{bbl@ADJ@layout.lists@off}{%
3595 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3596 %
3597 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3598 \bbl@bcpallowedtrue}
3599 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3600 \bbl@bcpallowedfalse}
3601 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3602 \def\bbl@bcp@prefix{#1}}
3603 \def\bbl@bcp@prefix{bcp47-}
3604 \@namedef{bbl@ADJ@autoload.options}#1{%
3605 \def\bbl@autoload@options{#1}}
3606 \def\bbl@autoload@bcptoptions{import}
3607 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3608 \def\bbl@autoload@bcptoptions{#1}}
3609 \newif\ifbbl@bcptname
3610 %
3611 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3612 \bbl@bcptnametrue}
3613 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3614 \bbl@bcptnamefalse}
3615 %
3616 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3617 \directlua{ Babel.ignore_pre_char = function(node)
3618 return (node.lang == \the\csname l@nohyphenation\endcsname)
3619 end }}
3620 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3621 \directlua{ Babel.ignore_pre_char = function(node)
3622 return false
3623 end }}
3624 %
3625 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3626 \def\bbl@ignoreinterchar{%
3627 \ifnum\language=\l@nohyphenation
3628 \expandafter\@gobble

```



```

3629 \else
3630 \expandafter\@firstofone
3631 \fi}}
3632 \@namedef{bbl@ADJ@interchar.disable@off}{%
3633 \let\bbl@ignoreinterchar\@firstofone}
3634 %
3635 \@namedef{bbl@ADJ@select.write@shift}{%
3636 \let\bbl@restorelastskip\relax
3637 \def\bbl@savelastskip{%
3638 \let\bbl@restorelastskip\relax
3639 \ifvmode
3640 \ifdim\lastskip=\z@
3641 \let\bbl@restorelastskip\nobreak
3642 \else
3643 \bbl@exp{%
3644 \def\\bbl@restorelastskip{%
3645 \skip@=\the\lastskip
3646 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3647 \fi
3648 \fi}}
3649 \@namedef{bbl@ADJ@select.write@keep}{%
3650 \let\bbl@restorelastskip\relax
3651 \let\bbl@savelastskip\relax}
3652 \@namedef{bbl@ADJ@select.write@omit}{%
3653 \AddBabelHook{babel-select}{beforestart}{%
3654 \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3655 \let\bbl@restorelastskip\relax
3656 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3657 \@namedef{bbl@ADJ@select.encoding@off}{%
3658 \let\bbl@encoding@select@off\@empty}

```

5.1. Cross referencing macros

The \TeX 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.

```

3659 <<(*More package options)>> ≡
3660 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3661 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3662 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3663 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3664 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3665 <</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.

```

3666 \bbl@trace{Cross referencing macros}
3667 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3668 \def\@newl@bel#1#2#3{%
3669 {\@safe@activestrue
3670 \bbl@ifunset{#1@#2}%
3671 \relax
3672 {\gdef\@multiplelabels{%
3673 \@latex@warning@no@line{There were multiply-defined labels}}%

```

```

3674 \latex@warning@no@line{Label `#2' multiply defined}}%
3675 \global\@namedef{#1@#2}{#3}}

```

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

```

3676 \CheckCommand*\@testdef[3]{%
3677 \def\reserved@a{#3}%
3678 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3679 \else
3680 \@tempswatrue
3681 \fi}

```

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

```

3682 \def\@testdef#1#2#3{%
3683 \@safe@activestrue
3684 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3685 \def\bbl@tempb{#3}%
3686 \@safe@activestfalse
3687 \ifx\bbl@tempa\relax
3688 \else
3689 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3690 \fi
3691 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3692 \ifx\bbl@tempa\bbl@tempb
3693 \else
3694 \@tempswatrue
3695 \fi}
3696 \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.

```

3697 \bbl@xin@{R}\bbl@opt@safe
3698 \ifin@
3699 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3700 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3701 {\expandafter\strip@prefix\meaning\ref}%
3702 \ifin@
3703 \bbl@redefine\@kernel@ref#1{%
3704 \@safe@activestrue\org@@kernel@ref{#1}\@safe@activestfalse}
3705 \bbl@redefine\@kernel@pageref#1{%
3706 \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activestfalse}
3707 \bbl@redefine\@kernel@sref#1{%
3708 \@safe@activestrue\org@@kernel@sref{#1}\@safe@activestfalse}
3709 \bbl@redefine\@kernel@spageref#1{%
3710 \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activestfalse}
3711 \else
3712 \bbl@redefineroobust\ref#1{%
3713 \@safe@activestrue\org@ref{#1}\@safe@activestfalse}
3714 \bbl@redefineroobust\pageref#1{%
3715 \@safe@activestrue\org@pageref{#1}\@safe@activestfalse}
3716 \fi
3717 \else
3718 \let\org@ref\ref
3719 \let\org@pageref\pageref
3720 \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.

```
3721 \bbl@xin@{B}\bbl@opt@safe
3722 \ifin@
3723 \bbl@redefine\@citex[#1]#2{%
3724   \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetruefalse
3725   \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.)

```
3726 \AtBeginDocument{%
3727   \ifpackageloaded{natbib}{%
3728     \def\@citex[#1][#2]#3{%
3729       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activetruefalse
3730       \org@citex[#1][#2]{\bbl@tempa}}%
3731   }}
```

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

```
3732 \AtBeginDocument{%
3733   \ifpackageloaded{cite}{%
3734     \def\@citex[#1]#2{%
3735       \@safe@activetrue\org@citex[#1]{#2}\@safe@activetruefalse}%
3736   }}
```

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

```
3737 \bbl@redefine\nocite#1{%
3738   \@safe@activetrue\org@nocite{#1}\@safe@activetruefalse}
```

\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.

```
3739 \bbl@redefine\bibcite{%
3740   \bbl@cite@choice
3741   \bibcite}
```

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

```
3742 \def\bbl@bibcite#1#2{%
3743   \org@bibcite{#1}{\@safe@activetruefalse#2}}
```

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

```
3744 \def\bbl@cite@choice{%
3745   \global\let\bibcite\bbl@bibcite
3746   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3747   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3748   \global\let\bbl@cite@choice\relax}
```

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

```
3749 \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.

```
3750 \bbl@redefine\@bibitem#1{%
3751   \@safe@activestruelorg@@bibitem{#1}\@safe@activesfalse}
3752 \else
3753   \let\org@nocite\nocite
3754   \let\org@@citex\@citex
3755   \let\org@babcite\babcite
3756   \let\org@@bibitem\@bibitem
3757 \fi
```

5.2. Layout

```
3758 \newcommand\BabelPatchSection[1]{%
3759   \@ifundefined{#1}{}{%
3760     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3761     \@namedef{#1}{%
3762       \@ifstar{\bbl@presec@s{#1}}%
3763       {\@dblarg{\bbl@presec@x{#1}}}}}%
3764 \def\bbl@presec@x#1[#2]#3{%
3765   \bbl@exp{%
3766     \\\select@language@x{\bbl@main@language}%
3767     \\\bbl@cs{sspre@#1}%
3768     \\\bbl@cs{ss@#1}%
3769     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3770     {\\\foreignlanguage{\language}{\unexpanded{#3}}}%
3771     \\\select@language@x{\language}}}%
3772 \def\bbl@presec@s#1#2{%
3773   \bbl@exp{%
3774     \\\select@language@x{\bbl@main@language}%
3775     \\\bbl@cs{sspre@#1}%
3776     \\\bbl@cs{ss@#1}*%
3777     {\\\foreignlanguage{\language}{\unexpanded{#2}}}%
3778     \\\select@language@x{\language}}}%
3779 %
3780 \IfBabelLayout{sectioning}%
3781   {\BabelPatchSection{part}%
3782    \BabelPatchSection{chapter}%
3783    \BabelPatchSection{section}%
3784    \BabelPatchSection{subsection}%
3785    \BabelPatchSection{subsubsection}%
3786    \BabelPatchSection{paragraph}%
3787    \BabelPatchSection{subparagraph}%
3788    \def\babel@toc#1{%
3789      \select@language@x{\bbl@main@language}}}%
3790 \IfBabelLayout{captions}%
3791   {\BabelPatchSection{caption}}{}}
```

\BabelFootnote Footnotes.

```
3792 \bbl@trace{Footnotes}
3793 \def\bbl@footnote#1#2#3{%
3794   \@ifnextchar[%
3795     {\bbl@footnote@o{#1}{#2}{#3}}%
3796     {\bbl@footnote@x{#1}{#2}{#3}}}
3797 \long\def\bbl@footnote@x#1#2#3#4{%
3798   \bgroup
3799   \select@language@x{\bbl@main@language}%
3800   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%

```

```

3801 \egroup}
3802 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3803 \bgroup
3804 \select@language@x{\bbl@main@language}%
3805 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3806 \egroup}
3807 \def\bbl@footnotetext#1#2#3{%
3808 \@ifnextchar[%
3809 {\bbl@footnotetext@o{#1}{#2}{#3}}%
3810 {\bbl@footnotetext@x{#1}{#2}{#3}}}
3811 \long\def\bbl@footnotetext@x#1#2#3#4{%
3812 \bgroup
3813 \select@language@x{\bbl@main@language}%
3814 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3815 \egroup}
3816 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3817 \bgroup
3818 \select@language@x{\bbl@main@language}%
3819 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3820 \egroup}
3821 \def\BabelFootnote#1#2#3#4{%
3822 \ifx\bbl@fn@footnote\@undefined
3823 \let\bbl@fn@footnote\footnote
3824 \fi
3825 \ifx\bbl@fn@footnotetext\@undefined
3826 \let\bbl@fn@footnotetext\footnotetext
3827 \fi
3828 \bbl@ifblank{#2}%
3829 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3830 \namedef{\bbl@stripslash#1text}%
3831 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3832 {\def#1{\bbl@exp{\bbl@footnote{\bbl@foreignlanguage{#2}}}{#3}{#4}}%
3833 \namedef{\bbl@stripslash#1text}%
3834 {\bbl@exp{\bbl@footnotetext{\bbl@foreignlanguage{#2}}}{#3}{#4}}}%
3835 \IfBabelLayout{footnotes}%
3836 {\let\bbl@OL@footnote\footnote
3837 \BabelFootnote\footnote\language\name{}}}%
3838 \BabelFootnote\localfootnote\language\name{}}}%
3839 \BabelFootnote\mainfootnote{}}}%
3840 {}

```

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.

```

3841 \bbl@trace{Marks}
3842 \IfBabelLayout{sectioning}
3843 {\ifx\bbl@opt@headfoot\@nnil
3844 \g@addto@macro\@resetactivechars{%
3845 \set@typeset@protect
3846 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3847 \let\protect\noexpand
3848 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3849 \edef\thepage{%
3850 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3851 \fi}%
3852 \fi}
3853 {\ifbbl@single\else
3854 \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust

```

```

3855 \markright#1{%
3856 \bbl@ifblank{#1}%
3857 {\org@markright{}}}%
3858 {\toks@{#1}%
3859 \bbl@exp{%
3860 \\\org@markright{\\protect\\foreignlanguage{\\language}%
3861 {\\protect\\bbl@restore@actives\the\toks@}}}%

```

\markboth

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

```

3862 \ifx\@mkboth\markboth
3863 \def\bbl@tempc{\let\@mkboth\markboth}%
3864 \else
3865 \def\bbl@tempc{%
3866 \fi
3867 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3868 \markboth#1#2{%
3869 \protected@edef\bbl@tempb##1{%
3870 \protect\foreignlanguage
3871 {\\language}%{\protect\bbl@restore@actives##1}}%
3872 \bbl@ifblank{#1}%
3873 {\toks@{}}%
3874 {\toks@\expandafter{\bbl@tempb{#1}}}%
3875 \bbl@ifblank{#2}%
3876 {\@temptokena{}}%
3877 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3878 \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3879 \bbl@tempc
3880 \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.

```

3881 \bbl@trace{Preventing clashes with other packages}
3882 \ifx\org@ref\undefined\else
3883 \bbl@xin@{R}\bbl@opt@safe
3884 \ifin@
3885 \AtBeginDocument{%
3886 \@ifpackageloaded{ifthen}{%
3887 \bbl@redefine@long\ifthenelse#1#2#3{%

```

```

3888      \let\bbl@temp@pref\pageref
3889      \let\pageref\org@pageref
3890      \let\bbl@temp@ref\ref
3891      \let\ref\org@ref
3892      \@safe@activetrue
3893      \org@ifthenelse{#1}%
3894      {\let\pageref\bbl@temp@pref
3895       \let\ref\bbl@temp@ref
3896       \@safe@activesfalse
3897       #2}%
3898      {\let\pageref\bbl@temp@pref
3899       \let\ref\bbl@temp@ref
3900       \@safe@activesfalse
3901       #3}%
3902      }%
3903    }{}%
3904  }
3905 \fi

```

5.4.2. varioref

\@@vpageref

\vrefpagemum

\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 \vrefpagemum.

```

3906 \AtBeginDocument{%
3907   \ifpackageloaded{varioref}{%
3908     \bbl@redefine\@@vpageref#1[#2]#3{%
3909       \@safe@activetrue
3910       \org@@@vpageref{#1}[#2]{#3}%
3911       \@safe@activesfalse}%
3912     \bbl@redefine\vrefpagemum#1#2{%
3913       \@safe@activetrue
3914       \org@vrefpagemum{#1}#2}%
3915     \@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.

```

3916   \expandafter\def\csname Ref \endcsname#1{%
3917     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3918   }{}%
3919 }
3920 \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.

```

3921 \AtEndOfPackage{%
3922   \AtBeginDocument{%
3923     \ifpackageloaded{hhline}%
3924     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3925      \else
3926        \makeatletter
3927        \def\@currname{hhline}\input{hhline.sty}\makeatother

```

```

3928     \fi}%
3929     {}}}

```

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

```

3930 \def\substitutefontfamily#1#2#3{%
3931   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3932   \immediate\writel5{%
3933     \string\ProvidesFile{#1#2.fd}%
3934     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3935     \space generated font description file]^J
3936     \string\DeclareFontFamily{#1}{#2}{}}^J
3937     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3938     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3939     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3940     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3941     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3942     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3943     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3944     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3945   }%
3946   \closeout15
3947 }
3948 \@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 $\text{\@fontenc@load@list}$. 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

```

3949 \bbl@trace{Encoding and fonts}
3950 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3951 \newcommand\BabelNonText{TS1,T3,TS3}
3952 \let\org@TeX\TeX
3953 \let\org@LaTeX\LaTeX
3954 \let\ensureascii@firstofone
3955 \let\asciientcoding@empty
3956 \AtBeginDocument{%
3957   \def\@elt#1{,#1,}%
3958   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3959   \let\@elt\relax
3960   \let\bbl@tempb@empty
3961   \def\bbl@tempc{OT1}%
3962   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3963     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3964   \bbl@foreach\bbl@tempa{%
3965     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3966     \ifin@
3967       \def\bbl@tempb{#1}% Store last non-ascii
3968     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3969       \ifin@else
3970       \def\bbl@tempc{#1}% Store last ascii
3971     \fi
3972     \fi}%
3973   \ifx\bbl@tempb@empty\else
3974     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3975     \ifin@else

```



```

3976     \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3977     \fi
3978     \let\asciencoding\bbl@tempc
3979     \renewcommand\ensureascii[1]{%
3980       {\fontencoding{\asciencoding}\selectfont#1}}%
3981     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3982     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3983     \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.

```

3984 \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.

```

3985 \AtBeginDocument{%
3986   \ifpackageloaded{fontspec}%
3987     {\xdef\latinencoding{%
3988       \ifx\UTFencname\@undefined
3989         EU\ifcase\bbl@engine\or2\or1\fi
3990       \else
3991         \UTFencname
3992       \fi}}%
3993   {\gdef\latinencoding{OT1}%
3994     \ifx\cf@encoding\bbl@t@one
3995       \xdef\latinencoding{\bbl@t@one}%
3996     \else
3997       \def\@elt#1{, #1,}%
3998       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3999       \let\@elt\relax
4000       \bbl@xin@{, T1, }\bbl@tempa
4001       \ifin@
4002         \xdef\latinencoding{\bbl@t@one}%
4003       \fi
4004     \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.

```

4005 \DeclareRobustCommand{\latintext}{%
4006   \fontencoding{\latinencoding}\selectfont
4007   \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.

```

4008 \ifx\@undefined\DeclareTextFontCommand
4009   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
4010 \else
4011   \DeclareTextFontCommand{\textlatin}{\latintext}
4012 \fi

```

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

```

4013 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.6. Basic bidi support

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

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

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

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

```
4014 \bbl@trace{Loading basic (internal) bidi support}
4015 \ifodd\bbl@engine
4016 \else % Any xe+lua bidi
4017   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
4018     \bbl@error{bidi-only-lua}{}}{}%
4019     \let\bbl@beforeforeign\leavevmode
4020     \AtEndOfPackage{%
4021       \EnableBabelHook{babel-bidi}%
4022       \bbl@xebidipar}
4023   \fi\fi
4024   \def\bbl@loadxebidi#1{%
4025     \ifx\RTLfootnotetext\@undefined
4026       \AtEndOfPackage{%
4027         \EnableBabelHook{babel-bidi}%
4028         \ifx\fontspec\@undefined
4029           \usepackage{fontspec}% bidi needs fontspec
4030         \fi
4031         \usepackage#1{bidi}%
4032         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4033         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4034           \ifnum\@nameuse{\bbl@wdir\@languagename}=\tw@ % 'AL' bidi
4035             \bbl@digitsdotdash % So ignore in 'R' bidi
4036           \fi}}%
4037     \fi}
4038   \ifnum\bbl@bidimode>200 % Any xe bidi=
4039     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4040       \bbl@tentative{bidi=bidi}
4041       \bbl@loadxebidi{}
4042     \or
4043       \bbl@loadxebidi{[rldocument]}
4044     \or
4045       \bbl@loadxebidi{}
4046     \fi
4047   \fi
4048 \fi
4049 \ifnum\bbl@bidimode=\@ne % bidi=default
4050   \let\bbl@beforeforeign\leavevmode
4051   \ifodd\bbl@engine % lua
4052     \newattribute\bbl@attr@dir
4053     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4054     \bbl@exp{\output{\bodydir\pagedir\the\output}}
```

```

4055 \fi
4056 \AtEndOfPackage{%
4057   \EnableBabelHook{babel-bidi}% pdf/luax/xe
4058   \ifodd\bbbl@engine\else % pdf/xe
4059     \bbl@xebidipar
4060   \fi}
4061 \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.

```

4062 \bbl@trace{Macros to switch the text direction}
4063 \def\bbl@alscripts{%
4064   ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
4065 \def\bbl@rscripts{%
4066   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4067   Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
4068   Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaean,%
4069   Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
4070   Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
4071   Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
4072   Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
4073   Meroitic,N'Ko,Orkhon,Todhri}
4074 %
4075 \def\bbl@provide@dirs#1{%
4076   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4077   \ifin@
4078     \global\bbl@csarg\chardef{wdir@#1}\@ne
4079     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4080     \ifin@
4081       \global\bbl@csarg\chardef{wdir@#1}\tw@
4082     \fi
4083   \else
4084     \global\bbl@csarg\chardef{wdir@#1}\z@
4085   \fi
4086   \ifodd\bbl@engine
4087     \bbl@csarg\ifcase{wdir@#1}%
4088       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4089     \or
4090       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4091     \or
4092       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4093     \fi
4094   \fi}
4095 %
4096 \def\bbl@switchdir{%
4097   \bbl@ifunset{\bbl@sys@\language name}{\bbl@provide@sys@\language name}}{%
4098   \bbl@ifunset{\bbl@wdir@\language name}{\bbl@provide@dirs@\language name}}{%
4099   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4100 \def\bbl@setdirs#1{%
4101   \ifcase\bbl@select@type
4102     \bbl@bodydir{#1}%
4103     \bbl@pardir{#1}% <- Must precede \bbl@textdir
4104   \fi
4105   \bbl@textdir{#1}}
4106 \ifnum\bbl@bidimode>\z@
4107   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4108   \DisableBabelHook{babel-bidi}
4109 \fi

```

Now the engine-dependent macros.

```

4110 \ifodd\bbl@engine % luatex=1
4111 \else % pdftex=0, xetex=2
4112   \newcount\bbl@dirlevel

```

```

4113 \chardef\bbl@thetextdir\z@
4114 \chardef\bbl@thepardir\z@
4115 \def\bbl@textdir#1{%
4116   \ifcase#1\relax
4117     \chardef\bbl@thetextdir\z@
4118     \@nameuse{setlatin}%
4119     \bbl@textdir@i\beginL\endL
4120   \else
4121     \chardef\bbl@thetextdir@ne
4122     \@nameuse{setnonlatin}%
4123     \bbl@textdir@i\beginR\endR
4124   \fi}
4125 \def\bbl@textdir@i#1#2{%
4126   \ifhmode
4127     \ifnum\currentgrouplevel>\z@
4128       \ifnum\currentgrouplevel=\bbl@dirlevel
4129         \bbl@error{multiple-bidi}{\}\}\}%
4130         \bgroup\aftergroup#2\aftergroup\egroup
4131       \else
4132         \ifcase\currentgrouptype\or % 0 bottom
4133           \aftergroup#2% 1 simple {}
4134         \or
4135           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4136         \or
4137           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4138         \or\or\or % vbox vtop align
4139         \or
4140           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4141         \or\or\or\or\or\or % output math disc insert vcent mathchoice
4142         \or
4143           \aftergroup#2% 14 \begingroup
4144         \else
4145           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4146         \fi
4147       \fi
4148       \bbl@dirlevel\currentgrouplevel
4149     \fi
4150   #1%
4151 \fi}
4152 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4153 \let\bbl@bodydir@gobble
4154 \let\bbl@pagedir@gobble
4155 \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).

```

4156 \def\bbl@xebidipar{%
4157   \let\bbl@xebidipar\relax
4158   \TeXeTstate@ne
4159   \def\bbl@xeeverypar{%
4160     \ifcase\bbl@thepardir
4161       \ifcase\bbl@thetextdir\else\beginR\fi
4162     \else
4163       {\setbox\z@\lastbox\beginR\box\z@}%
4164     \fi}%
4165   \AddToHook{para/begin}{\bbl@xeeverypar}}
4166 \ifnum\bbl@bidimode>200 % Any xe bidi=
4167   \let\bbl@textdir@i@gobbletwo
4168   \let\bbl@xebidipar@empty
4169   \AddBabelHook{bidi}{foreign}{%
4170     \ifcase\bbl@thetextdir
4171       \BabelWrapText{\LR{##1}}}%

```

```

4172     \else
4173     \BabelWrapText{\RL{##1}}%
4174     \fi}
4175 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4176 \fi
4177 \fi

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

4178 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4179 \AtBeginDocument{%
4180 \ifx\pdfstringdefDisableCommands\@undefined\else
4181 \ifx\pdfstringdefDisableCommands\relax\else
4182 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4183 \fi
4184 \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.

```

4185 \bbl@trace{Local Language Configuration}
4186 \ifx\loadlocalcfg\@undefined
4187 \ifpackagewith{babel}{noconfigs}%
4188 {\let\loadlocalcfg\@gobble}%
4189 {\def\loadlocalcfg#1{%
4190 \InputIfFileExists{#1.cfg}%
4191 {\typeout{*****^J%
4192 * Local config file #1.cfg used^^J%
4193 *}}%
4194 \@empty}}
4195 \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).

```

4196 \bbl@trace{Language options}
4197 \def\BabelDefinitionFile#1#2#3{}
4198 \let\bbl@afterlang\relax
4199 \let\BabelModifiers\relax
4200 \let\bbl@loaded\@empty
4201 \def\bbl@load@language#1{%
4202 \InputIfFileExists{#1.ldf}%
4203 {\edef\bbl@loaded{\CurrentOption
4204 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4205 \expandafter\let\expandafter\bbl@afterlang
4206 \csname\CurrentOption.ldf-h@@k\endcsname
4207 \expandafter\let\expandafter\BabelModifiers
4208 \csname bbl@mod@\CurrentOption\endcsname
4209 \bbl@exp{\AtBeginDocument{%
4210 \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4211 {\bbl@error{unknown-package-option}{}}}}

```

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

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

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

```

4212 \ifx\GetDocumentProperties\@undefined\else
4213   \let\bbl@beforeforeign\leavevmode
4214   \edef\bbl@metalang{\GetDocumentProperties{document/lang}}%
4215   \ifx\bbl@metalang\empty\else
4216     \begingroup
4217       \expandafter
4218       \bbl@bcpllookup\bbl@metalang-\@empty-\@empty-\@empty@@
4219       \ifx\bbl@bcplrelax
4220         \ifx\bbl@opt@main\@nnil
4221           \bbl@error{no-locale-for-meta}{\bbl@metalang}{}{}%
4222         \fi
4223       \else
4224         \bbl@read@ini{\bbl@bcpl}\m@ne
4225         \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4226         \ifx\bbl@opt@main\@nnil
4227           \global\let\bbl@opt@main\language
4228         \fi
4229         \bbl@info{Passing \language\space to babel.\\%
4230                  This will be the main language except if\\%
4231                  explicitly overridden with 'main='.\\%
4232                  Reported}%
4233       \fi
4234     \endgroup
4235   \fi
4236 \fi
4237 \ifx\bbl@opt@config\@nnil
4238   \@ifpackagewith{babel}{noconfigs}{}%
4239   {\InputIfFileExists{bblopts.cfg}%
4240    {\bbl@info{Configuration files are deprecated, as\\%
4241              they can break document portability.\\%
4242              Reported}%
4243     \typeout{*****^J%
4244              * Local config file bblopts.cfg used^^J%
4245              *}}%
4246    {}}%
4247 \else
4248   \InputIfFileExists{\bbl@opt@config.cfg}%
4249   {\bbl@info{Configuration files are deprecated, as\\%
4250             they can break document portability.\\%
4251             Reported}%
4252    \typeout{*****^J%
4253             * Local config file \bbl@opt@config.cfg used^^J%
4254             *}}%
4255   {\bbl@error{config-not-found}{}{}{}%
4256 \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).

```

4257 \def\BabelBeforeIni#1#2{%
4258   \def\bbl@tempa{\@m}% <- Default if no \BDefFile
4259   \let\bbl@tempb\@empty
4260   #2%
4261   \edef\bbl@toload{%
4262     \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4263     \bbl@toload@last}%
4264   \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4265 \def\BabelDefinitionFile#1#2#3{%
4266   \def\bbl@tempa{#1}\def\bbl@tempb{#2}%
4267   \@namedef{bbl@preldf@\CurrentOption}{#3}%
4268   \endinput}%

```

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

```

4269 \def\bbl@tempf{,}
4270 \bbl@foreach\@raw@classoptionslist{%
4271   \in@{=}{#1}%
4272   \ifin@else
4273     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4274   \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.

```

4275 \let\bbl@toload\@empty
4276 \let\bbl@toload@last\@empty
4277 \let\bbl@unkopt\@gobble %% <- Ugly
4278 \edef\bbl@tempc{%
4279   \bbl@tempf,@@,\bbl@language@opts
4280   \ifx\bbl@opt@main\@nnil\else,\bbl@opt@main\fi}
4281 \let\BabelLocalesTentative\bbl@tempc
4282 %
4283 \bbl@foreach\bbl@tempc{%
4284   \in@{@@}{#1}% <- Ugly
4285   \ifin@
4286     \def\bbl@unkopt##1{%
4287       \DeclareOption{##1}{\bbl@error{unknown-package-option}}{}}}%
4288   \else
4289     \def\CurrentOption{#1}%
4290     \bbl@xin@{/#1//}{\bbl@toload@last}% Collapse consecutive
4291     \ifin@else
4292       \lowercase{\InputIfFileExists{babel-#1.tex}}{ }{%
4293         \IfFileExists{#1.ldf}%
4294         {\edef\bbl@toload{%
4295           \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4296           \bbl@toload@last}%
4297         \edef\bbl@toload@last{0/0//\CurrentOption//und/#1}}%
4298         {\bbl@unkopt{#1}}}%
4299     \fi
4300   \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.

```

4301 \ifx\bbl@opt@main\@nnil
4302   \ifx\bbl@toload@last\@empty
4303     \def\bbl@toload@last{0/0//nil//und-x-nil/nil}
4304     \bbl@info{%

```

```

4305     You haven't specified a language as a class or package\\%
4306     option. I'll load 'nil'. Reported}
4307 \fi
4308 \else
4309 \let\bbl@tempc\@empty
4310 \bbl@foreach\bbl@toload{%
4311 \bbl@xin@{//\bbl@opt@main//}{#1}%
4312 \ifin@else
4313 \bbl@add@list\bbl@tempc{#1}%
4314 \fi}
4315 \let\bbl@toload\bbl@tempc
4316 \fi
4317 \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’.

```

4318 \def\AfterBabelLanguage#1{%
4319 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4320 \NewHook{babel/presets}
4321 \UseHook{babel/presets}
4322 %
4323 \let\bbl@tempb\@empty
4324 \def\bbl@tempc#1/#2/#3/#4/#5\@@{%
4325 \count@z@
4326 \ifnum#2=\@m % if no \BabelDefinitionFile
4327 \ifnum#1=z@ % not main. -- % if provide+=!, provide*=!
4328 \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0/#3/#4/#3\@@
4329 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4330 \fi
4331 \else % 10 = main -- % if provide+=!, provide*=!
4332 \ifodd\bbl@ldfflag\bbl@tempc 10/0/#3/#4/#3\@@
4333 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4334 \fi
4335 \fi
4336 \else
4337 \ifnum#1=z@ % not main
4338 \ifnum\bbl@iniflag>\@ne\else % if 0, provide
4339 \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4340 \fi
4341 \else % 10 = main
4342 \ifodd\bbl@iniflag\else % if provide+, provide*
4343 \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4344 \fi
4345 \fi
4346 \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4347 \fi}

```

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

```

4348 \def\bbl@tempd#1#2#3#4#5{%
4349 \DeclareOption{#3}{%
4350 \ifcase\count@
4351 \bbl@exp{\bbl@add\bbl@tempb{%
4352 \\\@nameuse\bbl@preini{#3}%
4353 \bbl@ldfinit
4354 \def\CurrentOption{#3}%
4355 \\\babelprovide[@import=#4,\ifnum#1=z@\else\bbl@opt@provide,main\fi]{#3}%
4356 \bbl@afterldf}}%
4357 \else
4358 \bbl@add\bbl@tempb{%
4359 \def\CurrentOption{#3}%
4360 \let\localename\CurrentOption

```



```

4361 \let\language\localename
4362 \def\BabelIniTag{#4}%
4363 \@nameuse{bbl@preldf@#3}%
4364 \begingroup
4365 \bbl@id@assign
4366 \bbl@read@ini{\BabelIniTag}0%
4367 \endgroup
4368 \bbl@load@language{#5}}%
4369 \fi}
4370 %
4371 \bbl@foreach\bbl@toload{\bbl@tempc#1\@@}
4372 \bbl@tempb
4373 \DeclareOption*{}
4374 \ProcessOptions
4375 %
4376 \bbl@exp{%
4377 \\\AtBeginDocument{\bbl@usehooks@lang{/}{\begindocument}}{}}}%
4378 \def\AfterBabelLanguage{\bbl@error{late-after-babel}}{}}{}
4379 </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

```
4380 <*/kernel>
4381 \let\bbl@onlyswitch\@empty
4382 \input babel.def
4383 \let\bbl@onlyswitch\@undefined
4384 </kernel>
```

7. Error messages

They are loaded when `\bl@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.

```

4385 (*errors)
4386 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4387 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4388 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4389 \catcode`\@=11 \catcode`\^=7
4390 %
4391 \ifx\MessageBreak\undefined
4392 \gdef\bbl@error@i#1#2{%
4393 \begingroup
4394 \newlinechar=^^J
4395 \def\{^^J(babel) }%
4396 \errhelp{#2}\errmessage{\{#1}%
4397 \endgroup}
4398 \else
4399 \gdef\bbl@error@i#1#2{%
4400 \begingroup

```

```

4401 \def\{\MessageBreak}%
4402 \PackageError{babel}{#1}{#2}%
4403 \endgroup}
4404 \fi
4405 \def\bbl@errmessage#1#2#3{%
4406 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4407 \bbl@error@i{#2}{#3}}
4408 % Implicit #2#3#4:
4409 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4410 %
4411 \bbl@errmessage{not-yet-available}
4412 {Not yet available}%
4413 {Find an armchair, sit down and wait}
4414 \bbl@errmessage{bad-package-option}%
4415 {Bad option '#1=#2'. Either you have misspelled the\\%
4416 key or there is a previous setting of '#1'. Valid\\%
4417 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4418 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4419 {See the manual for further details.}
4420 \bbl@errmessage{base-on-the-fly}
4421 {For a language to be defined on the fly 'base'\\%
4422 is not enough, and the whole package must be\\%
4423 loaded. Either delete the 'base' option or\\%
4424 request the languages explicitly}%
4425 {See the manual for further details.}
4426 \bbl@errmessage{undefined-language}
4427 {You haven't defined the language '#1' yet.\\%
4428 Perhaps you misspelled it or your installation\\%
4429 is not complete}%
4430 {Your command will be ignored, type <return> to proceed}
4431 \bbl@errmessage{invalid-ini-name}
4432 {'#1' not valid with the 'ini' mechanism.\\%
4433 I think you want '#2' instead. You may continue,\\%
4434 but you should fix the name. See the babel manual\\%
4435 for the available locales with 'provide'}%
4436 {See the manual for further details.}
4437 \bbl@errmessage{shorthand-is-off}
4438 {I can't declare a shorthand turned off (\string#2)}
4439 {Sorry, but you can't use shorthands which have been\\%
4440 turned off in the package options}
4441 \bbl@errmessage{not-a-shorthand}
4442 {The character '\string #1' should be made a shorthand character;\\%
4443 add the command \string\usesshorthands\string{#1\string} to
4444 the preamble.\\%
4445 I will ignore your instruction}%
4446 {You may proceed, but expect unexpected results}
4447 \bbl@errmessage{not-a-shorthand-b}
4448 {I can't switch '\string#2' on or off--not a shorthand\\%
4449 This character is not a shorthand. Maybe you made\\%
4450 a typing mistake?}%
4451 {I will ignore your instruction.}
4452 \bbl@errmessage{unknown-attribute}
4453 {The attribute #2 is unknown for language #1.}%
4454 {Your command will be ignored, type <return> to proceed}
4455 \bbl@errmessage{missing-group}
4456 {Missing group for string \string#1}%
4457 {You must assign strings to some category, typically\\%
4458 captions or extras, but you set none}
4459 \bbl@errmessage{only-lua-xe}
4460 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4461 {Consider switching to these engines.}
4462 \bbl@errmessage{only-lua}
4463 {This macro is available only in LuaLaTeX}%

```

```

4464 {Consider switching to that engine.}
4465 \bbl@errmessage{unknown-provide-key}
4466 {Unknown key '#1' in \string\babelprovide}%
4467 {See the manual for valid keys}%
4468 \bbl@errmessage{unknown-mapfont}
4469 {Option '\bbl@KVP@mapfont' unknown for\\%
4470 mapfont. Use 'direction'}%
4471 {See the manual for details.}
4472 \bbl@errmessage{no-ini-file}
4473 {There is no ini file for the requested language\\%
4474 (#1: \language). Perhaps you misspelled it or your\\%
4475 installation is not complete}%
4476 {Fix the name or reinstall babel.}
4477 \bbl@errmessage{digits-is-reserved}
4478 {The counter name 'digits' is reserved for mapping\\%
4479 decimal digits}%
4480 {Use another name.}
4481 \bbl@errmessage{limit-two-digits}
4482 {Currently two-digit years are restricted to the\\
4483 range 0-9999}%
4484 {There is little you can do. Sorry.}
4485 \bbl@errmessage{alphabetic-too-large}
4486 {Alphabetic numeral too large (#1)}%
4487 {Currently this is the limit.}
4488 \bbl@errmessage{no-ini-info}
4489 {I've found no info for the current locale.\\%
4490 The corresponding ini file has not been loaded\\%
4491 Perhaps it doesn't exist}%
4492 {See the manual for details.}
4493 \bbl@errmessage{unknown-ini-field}
4494 {Unknown field '#1' in \string\BCPdata.\\%
4495 Perhaps you misspelled it}%
4496 {See the manual for details.}
4497 \bbl@errmessage{unknown-locale-key}
4498 {Unknown key for locale '#2':\\%
4499 #3\\%
4500 \string#1 will be set to \string\relax}%
4501 {Perhaps you misspelled it.}%
4502 \bbl@errmessage{adjust-only-vertical}
4503 {Currently, #1 related features can be adjusted only\\%
4504 in the main vertical list}%
4505 {Maybe things change in the future, but this is what it is.}
4506 \bbl@errmessage{layout-only-vertical}
4507 {Currently, layout related features can be adjusted only\\%
4508 in vertical mode}%
4509 {Maybe things change in the future, but this is what it is.}
4510 \bbl@errmessage{bidi-only-lua}
4511 {The bidi method 'basic' is available only in\\%
4512 luatex. I'll continue with 'bidi=default', so\\%
4513 expect wrong results.\\%
4514 Suggested actions:\\%
4515 * If possible, switch to luatex, as xetex is not\\%
4516 recommend anymore.\\
4517 * If you can't, try 'bidi=bidi' with xetex.\\%
4518 * With pdftex, only 'bidi=default' is available.}%
4519 {See the manual for further details.}
4520 \bbl@errmessage{multiple-bidi}
4521 {Multiple bidi settings inside a group\\%
4522 I'll insert a new group, but expect wrong results.\\%
4523 Suggested action:\\%
4524 * Add a new group where appropriate.}
4525 {See the manual for further details.}
4526 \bbl@errmessage{unknown-package-option}

```

```

4527 {Unknown option '\CurrentOption'.\\%
4528 Suggested actions:\\%
4529 * Make sure you haven't misspelled it\\%
4530 * Check in the babel manual that it's supported\\%
4531 * If supported and it's a language, you may\\%
4532 \space\space need in some distributions a separate\\%
4533 \space\space installation\\%
4534 * If installed, check there isn't an old\\%
4535 \space\space version of the required files in your system}
4536 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4537 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4538 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4539 \bbl@errmessage{config-not-found}
4540 {Local config file '\bbl@opt@config.cfg' not found.\\%
4541 Suggested actions:\\%
4542 * Make sure you haven't misspelled it in config=\\%
4543 * Check it exists and it's in the correct path}%
4544 {Perhaps you misspelled it.}
4545 \bbl@errmessage{late-after-babel}
4546 {Too late for \string\AfterBabelLanguage}%
4547 {Languages have been loaded, so I can do nothing}
4548 \bbl@errmessage{double-hyphens-class}
4549 {Double hyphens aren't allowed in \string\babelcharclass\\%
4550 because it's potentially ambiguous}%
4551 {See the manual for further info}
4552 \bbl@errmessage{unknown-interchar}
4553 {'#1' for '\language' cannot be enabled.\\%
4554 Maybe there is a typo}%
4555 {See the manual for further details.}
4556 \bbl@errmessage{unknown-interchar-b}
4557 {'#1' for '\language' cannot be disabled.\\%
4558 Maybe there is a typo}%
4559 {See the manual for further details.}
4560 \bbl@errmessage{charproperty-only-vertical}
4561 {\string\babelcharproperty\space can be used only in\\%
4562 vertical mode (preamble or between paragraphs)}%
4563 {See the manual for further info}
4564 \bbl@errmessage{unknown-char-property}
4565 {No property named '#2'. Allowed values are\\%
4566 direction (bc), mirror (bmg), and linebreak (lb)}%
4567 {See the manual for further info}
4568 \bbl@errmessage{bad-transform-option}
4569 {Bad option '#1' in a transform.\\%
4570 I'll ignore it but expect more errors}%
4571 {See the manual for further info.}
4572 \bbl@errmessage{font-conflict-transforms}
4573 {Transforms cannot be re-assigned to different\\%
4574 fonts. The conflict is in '\bbl@kv@label'.\\%
4575 Apply the same fonts or use a different label}%
4576 {See the manual for further details.}
4577 \bbl@errmessage{transform-not-available}
4578 {'#1' for '\language' cannot be enabled.\\%
4579 Maybe there is a typo or it's a font-dependent transform}%
4580 {See the manual for further details.}
4581 \bbl@errmessage{transform-not-available-b}
4582 {'#1' for '\language' cannot be disabled.\\%
4583 Maybe there is a typo or it's a font-dependent transform}%
4584 {See the manual for further details.}
4585 \bbl@errmessage{year-out-range}
4586 {Year out of range.\\%
4587 The allowed range is #1}%
4588 {See the manual for further details.}
4589 \bbl@errmessage{only-pdftex-lang}

```

```

4590 {The '#1' ldf style doesn't work with #2,\%
4591 but you can use the ini locale instead.\%
4592 Try adding 'provide=*' to the option list. You may\%
4593 also want to set 'bidi=' to some value}%
4594 {See the manual for further details.}
4595 \bbl@errmessage{hyphenmins-args}
4596 {\string\babelhyphenmins\ accepts either the optional\%
4597 argument or the star, but not both at the same time}%
4598 {See the manual for further details.}
4599 \bbl@errmessage{no-locale-for-meta}
4600 {There isn't currently a locale for the 'lang' requested\%
4601 in the PDF metadata ('#1'). To fix it, you can\%
4602 set explicitly a similar language (using the same\%
4603 script) with the key main= when loading babel. If you\%
4604 continue, I'll fallback to the 'nil' language, with\%
4605 tag 'und' and script 'Latn', but expect a bad font\%
4606 rendering with other scripts. You may also need set\%
4607 explicitly captions and date, too}%
4608 {See the manual for further details.}
4609 </errors>
4610 <*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.

```

4611 <@Make sure ProvidesFile is defined@>
4612 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4613 \xdef\bbl@format{\jobname}
4614 \def\bbl@version{<@version@>}
4615 \def\bbl@date{<@date@>}
4616 \ifx\AtBeginDocument\undefined
4617 \def\@empty{}
4618 \fi
4619 <@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.

```

4620 \def\process@line#1#2 #3 #4 {%
4621 \ifx=#1%
4622 \process@synonym{#2}%
4623 \else
4624 \process@language{#1#2}{#3}{#4}%
4625 \fi
4626 \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.

```

4627 \toks@{}
4628 \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.

```

4629 \def\process@synonym#1{%
4630 \ifnum\last@language=\m@ne
4631 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%

```

```

4632 \else
4633   \expandafter\chardef\csname l@#1\endcsname\last@language
4634   \wlog{\string\l@#1=\string\language\the\last@language}%
4635   \expandafter\let\csname #1hyphenmins\endcsname
4636   \csname\language\hyphenmins\endcsname
4637   \let\bb@elt\relax
4638   \edef\bb@languages{\bb@languages\bb@elt{#1}{\the\last@language}{}}}%
4639 \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 `\bb@get@enc` extracts the font encoding from the language name and stores it in `\bb@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

Some pattern files contain changes to the `\lccode` `\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.)

`\bb@languages` saves a snapshot of the loaded languages in the form `\bb@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.

```

4640 \def\process@language#1#2#3{%
4641   \expandafter\addlanguage\csname l@#1\endcsname
4642   \expandafter\language\csname l@#1\endcsname
4643   \edef\language{#1}%
4644   \bb@hook@everylanguage{#1}%
4645   % > luatex
4646   \bb@get@enc#1::\@@@
4647   \begingroup
4648     \lefthyphenmin\m@ne
4649     \bb@hook@loadpatterns{#2}%
4650     % > luatex
4651     \ifnum\lefthyphenmin=\m@ne
4652     \else
4653       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4654         \the\lefthyphenmin\the\righthyphenmin}%
4655     \fi
4656   \endgroup
4657   \def\bb@tempa{#3}%
4658   \ifx\bb@tempa\@empty\else
4659     \bb@hook@loadexceptions{#3}%
4660     % > luatex
4661   \fi
4662   \let\bb@elt\relax
4663   \edef\bb@languages{\bb@languages\bb@elt{#1}{\the\language}{#2}{\bb@tempa}}%
4664   \bb@languages\bb@elt{#1}{\the\language}{#2}{\bb@tempa}}%
4665   \ifnum\the\language=\z@

```

```

4666 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4667 \set@hyphenmins\tw@thr@@\relax
4668 \else
4669 \expandafter\expandafter\expandafter\set@hyphenmins
4670 \csname #1hyphenmins\endcsname
4671 \fi
4672 \the\toks@
4673 \toks@{}%
4674 \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.

```

4675 \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.

```

4676 \def\bbl@hook@everylanguage#1{}
4677 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4678 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4679 \def\bbl@hook@loadkernel#1{%
4680 \def\addlanguage{\csname newlanguage\endcsname}%
4681 \def\adddialect##1##2{%
4682 \global\chardef##1##2\relax
4683 \wlog{\string##1 = a dialect from \string\language##2}}%
4684 \def\iflanguage##1{%
4685 \expandafter\ifx\csname l@##1\endcsname\relax
4686 \nolater{##1}%
4687 \else
4688 \ifnum\csname l@##1\endcsname=\language
4689 \expandafter\expandafter\expandafter\@firstoftwo
4690 \else
4691 \expandafter\expandafter\expandafter\@secondoftwo
4692 \fi
4693 \fi}%
4694 \def\providehyphenmins##1##2{%
4695 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4696 \@namedef{##1hyphenmins}{##2}%
4697 \fi}%
4698 \def\set@hyphenmins##1##2{%
4699 \lefthyphenmin##1\relax
4700 \righthyphenmin##2\relax}%
4701 \def\selectlanguage{%
4702 \errhelp{Selecting a language requires a package supporting it}%
4703 \errmessage{No multilingual package has been loaded}}%
4704 \let\foreignlanguage\selectlanguage
4705 \let\otherlanguage\selectlanguage
4706 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4707 \def\bbl@usehooks##1##2{%
4708 \def\setlocale{%
4709 \errhelp{Find an armchair, sit down and wait}%
4710 \errmessage{(babel) Not yet available}}%
4711 \let\uselocale\setlocale
4712 \let\locale\setlocale
4713 \let\selectlocale\setlocale
4714 \let\localename\setlocale
4715 \let\textlocale\setlocale
4716 \let\textlanguage\setlocale
4717 \let\languagetext\setlocale}
4718 \begingroup
4719 \def\AddBabelHook#1##2{%
4720 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax

```

```

4721     \def\next{\toks1}%
4722     \else
4723     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4724     \fi
4725     \next}
4726 \ifx\directlua\@undefined
4727 \ifx\XeTeXinputencoding\@undefined\else
4728     \input xebabel.def
4729     \fi
4730 \else
4731     \input luababel.def
4732     \fi
4733 \openin1 = babel-\bbl@format.cfg
4734 \ifeof1
4735 \else
4736     \input babel-\bbl@format.cfg\relax
4737     \fi
4738 \closein1
4739 \endgroup
4740 \bbl@hook@loadkernel{switch.def}

```

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

```

4741 \openin1 = language.dat

```

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

```

4742 \def\language{english}%
4743 \ifeof1
4744 \message{I couldn't find the file language.dat,\space
4745         I will try the file hyphen.tex}
4746 \input hyphen.tex\relax
4747 \chardef\l@english\z@
4748 \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`.

```

4749 \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.

```

4750 \loop
4751 \endlinechar@m@ne
4752 \read1 to \bbl@line
4753 \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.

```

4754 \if T\ifeof1\fi T\relax
4755 \ifx\bbl@line\empty\else
4756     \edef\bbl@line{\bbl@line\space\space\space}%
4757     \expandafter\process@line\bbl@line\relax
4758     \fi
4759 \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.

```

4760 \begingroup
4761 \def\bbl@elt#1#2#3#4{%
4762     \global\language=#2\relax

```



```

4763      \gdef\languagename{#1}%
4764      \def\bbl@elt##1##2##3##4{}}}%
4765      \bbl@languages
4766      \endgroup
4767 \fi
4768 \closein1

```

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

```

4769 \if/\the\toks@/\else
4770   \errhelp{language.dat loads no language, only synonyms}
4771   \errmessage{Orphan language synonym}
4772 \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.

```

4773 \let\bbl@line\@undefined
4774 \let\process@line\@undefined
4775 \let\process@synonym\@undefined
4776 \let\process@language\@undefined
4777 \let\bbl@get@enc\@undefined
4778 \let\bbl@hyph@enc\@undefined
4779 \let\bbl@tempa\@undefined
4780 \let\bbl@hook@loadkernel\@undefined
4781 \let\bbl@hook@everylanguage\@undefined
4782 \let\bbl@hook@loadpatterns\@undefined
4783 \let\bbl@hook@loadexceptions\@undefined
4784 </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`).

```

4785 <<*More package options>> ≡
4786 \chardef\bbl@bidimode\z@
4787 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4788 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4789 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4790 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4791 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4792 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4793 <</More package options>>

```

\bblfont 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`.

```

4794 <<*Font selection>> ≡
4795 \bbl@trace{Font handling with fontspec}
4796 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4797 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfonts}
4798 \DisableBabelHook{babel-fontspec}
4799 \@onlypreamble\bblfont
4800 \ifx\NewDocumentCommand\@undefined\else % Not plain
4801   \NewDocumentCommand\bblfont{0}{m0}{m0}{\}%
4802     \bbl@bblfont@o[#1]{#2}{#3,#5}{#4}}
4803 \fi
4804 \newcommand\bbl@bblfont@o[2][\% 1=langs/scripts 2=fam
4805   \ifx\fontspec\@undefined
4806     \usepackage{fontspec}%

```

```

4807 \fi
4808 \EnableBabelHook{babel-fontspec}%
4809 \edef\bbl@tempa{#1}%
4810 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4811 \bbl@bblfont}
4812 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4813 \bbl@ifunset{\bbl@tempb family}%
4814 {\bbl@providefam{\bbl@tempb}}%
4815 {}%
4816 % For the default font, just in case:
4817 \bbl@ifunset{\bbl@sys@language}{\bbl@provide@sys@language}}{%
4818 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4819 {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
4820 \bbl@exp{%
4821 \let<\bbl@tempb dflt@\language><\bbl@tempb dflt@>%
4822 \\\bbl@fontset<\bbl@tempb dflt@\language>%
4823 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4824 {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4825 \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:

```

4826 \def\bbl@providefam#1{%
4827 \bbl@exp{%
4828 \\\newcommand<#1default>{}% Just define it
4829 \\\bbl@add@list\\bbl@font@fams{#1}%
4830 \\\NewHook{#1family}%
4831 \\\DeclareRobustCommand<#1family>{%
4832 \\\not@math@alphabet<#1family>\relax
4833 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4834 \\\fontfamily<#1default>%
4835 \\\UseHook{#1family}%
4836 \\\selectfont}%
4837 \\\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.

```

4838 \def\bbl@nostdfont#1{%
4839 \bbl@once{nostdfam-f@family}%
4840 {\bbl@infowarn{The current font is not a babel standard family:\\%
4841 #1%
4842 \fontname\font\\%
4843 There is nothing intrinsically wrong, and you can\\%,
4844 ignore this message altogether if you do not need\\%,
4845 this font. If they are used in the document, be aware\\%,
4846 'babel' will not set Script and Language for it, so\\%,
4847 you may consider defining a new family with \string\babelfont.\\%
4848 See the manual for further details about \string\babelfont.
4849 Reported}}
4850 {}}%
4851 \gdef\bbl@switchfont{%
4852 \bbl@ifunset{\bbl@sys@language}{\bbl@provide@sys@language}}{%
4853 \bbl@exp{% e.g., Arabic -> arabic
4854 \lowercase{\edef\\bbl@tempa{\bbl@c{l{sname}}}}}%
4855 \bbl@foreach\bbl@font@fams{%
4856 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4857 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4858 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4859 {}% 123=F - nothing!
4860 {\bbl@exp{% 3=T - from generic
4861 \global\let<\bbl@##1dflt@\language>%
4862 \<\bbl@##1dflt@>}}}%
4863 {\bbl@exp{% 2=T - from script
4864 \global\let<\bbl@##1dflt@\language>%
4865 \<\bbl@##1dflt@*\bbl@tempa>}}}%

```

```

4866     {}}%                                l=T - language, already defined
4867 \def\bbl@tempa{\bbl@nostdfont{}}%
4868 \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4869   \bbl@ifunset{\bbl@##1dflt@{\language\name}}%
4870   {\bbl@cs{famrst@##1}%
4871    \global\bbl@csarg\let{famrst@##1}\relax}%
4872   {\bbl@exp{% order is relevant.
4873     \\bbl@add\\originalTeX{%
4874       \\bbl@font@rst{\bbl@cl{##1dflt}}%
4875       \<##1default>\<##1family>{##1}}%
4876     \\bbl@font@set{\bbl@##1dflt@{\language\name}% the main part!
4877       \<##1default>\<##1family>}}}%
4878 \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`.

```

4879 \ifx\f@family\undefined\else      % if latex
4880 \ifcase\bbl@engine                  % if pdftex
4881   \let\bbl@ckeckstdfonts\relax
4882 \else
4883   \def\bbl@ckeckstdfonts{%
4884     \begingroup
4885     \global\let\bbl@ckeckstdfonts\relax
4886     \let\bbl@tempa\empty
4887     \bbl@foreach\bbl@font@fams{%
4888       \bbl@ifunset{\bbl@##1dflt@}%
4889       {\@nameuse{##1family}%
4890        \bbl@csarg\gdef{WFF@f@family}{}% Flag
4891        \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4892          \space\space\fontname\font\\}%
4893        \bbl@csarg\xdef{##1dflt@}{\f@family}%
4894        \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4895       {}}%
4896   \ifx\bbl@tempa\empty\else
4897     \bbl@infowarn{The following font families will use the default\\%
4898       settings for all or some languages:\\%
4899       \bbl@tempa
4900       There is nothing intrinsically wrong with it, but\\%
4901       'babel' will no set Script and Language, which could\\%
4902       be relevant in some languages. If your document uses\\%
4903       these families, consider redefining them with \string\babelfont.\\%
4904       Reported}%
4905   \fi
4906 \endgroup}
4907 \fi
4908 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4909 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4910   \bbl@xin@{<>}{#1}%
4911   \ifin@
4912     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4913   \fi
4914   \bbl@exp{%
4915     \def\\#2{#1}%          'Unprotected' macros return prev values
                           e.g., \rmdefault{\bbl@rmdflt@lang}

```

```

4916   \\bbl@ifsamestring{#2}{\f@family}%
4917   {\#3%
4918   \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}}%
4919   \let\\bbl@tempa\relax}%
4920   {}}}

```

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).

```

4921 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4922   \let\bbl@tempe\bbl@mapselect
4923   \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4924   \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}{}}%
4925   \let\bbl@mapselect\relax
4926   \let\bbl@tempfam#4%          e.g., '\rmfamily', to be restored below
4927   \let#4\@empty              %      Make sure \renewfontfamily is valid
4928   \bbl@set@renderer
4929   \bbl@exp{%
4930     \let\\bbl@tempfam<\bbl@stripslash#4\space>% e.g., '\rmfamily '
4931     \<keys_if_exist:nf>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4932     {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4933     \<keys_if_exist:nf>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4934     {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4935     \\renewfontfamily\\#4%
4936     [\bbl@cl{lsys},% xetex removes unknown features :-(
4937     \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4938     #2]}{#3}% i.e., \bbl@exp{.}{#3}
4939   \bbl@unset@renderer
4940   \begingroup
4941     #4%
4942     \xdef#1{\f@family}%      e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4943   \endgroup
4944   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4945   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4946   \ifin@
4947     \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4948   \fi
4949   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4950   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4951   \ifin@
4952     \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4953   \fi
4954   \let#4\bbl@tempfam
4955   \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@tempfam
4956   \let\bbl@mapselect\bbl@tempe}%

```

`font@rst` and `famrst` are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```

4957 \def\bbl@font@rst#1#2#3#4{%
4958   \bbl@ccarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4959 \def\bbl@font@fams{rm,sf,tt}
4960 <</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.

```
4961 < *xetex >
4962 \def\BabelStringsDefault{unicode}
4963 \let\xebbl@stop\relax
4964 \AddBabelHook{xetex}{encodedcommands}{%
4965   \def\bbl@tempa{#1}%
4966   \ifx\bbl@tempa\@empty
4967     \XeTeXinputencoding"bytes"%
4968   \else
4969     \XeTeXinputencoding"#1"%
4970   \fi
4971   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4972 \AddBabelHook{xetex}{stopcommands}{%
4973   \xebbl@stop
4974   \let\xebbl@stop\relax}
4975 \def\bbl@input@classes{% Used in CJK intraspaces
4976   \input{load-unicode-xetex-classes.tex}%
4977   \let\bbl@input@classes\relax}
4978 \def\bbl@intraspace#1 #2 #3\@{
4979   \bbl@csarg\gdef{xeisp@\language}%
4980   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4981 \def\bbl@intrapenalty#1\@{
4982   \bbl@csarg\gdef{xeipn@\language}%
4983   {\XeTeXlinebreakpenalty #1\relax}}
4984 \def\bbl@provide@intraspace{%
4985   \bbl@xin@{/s}{\bbl@cl{lnbrk}}}%
4986   \ifin@else\bbl@xin@{/c}{\bbl@cl{lnbrk}}}\fi
4987   \ifin@
4988     \bbl@ifunset{bbl@intsp@\language}{}%
4989     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4990       \ifx\bbl@KVP@intraspace\@nnil
4991         \bbl@exp{%
4992           \\bbl@intraspace\bbl@cl{intsp}\\\@}%
4993         \fi
4994         \ifx\bbl@KVP@intrapenalty\@nnil
4995           \bbl@intrapenalty0\@
4996         \fi
4997         \fi
4998         \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4999           \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
5000         \fi
5001         \ifx\bbl@KVP@intrapenalty\@nnil\else
5002           \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5003         \fi
5004         \bbl@exp{%
5005           \\bbl@add\<extras\language>{%
5006             \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
5007             \<bbl@xeisp@\language>%
5008             \<bbl@xeipn@\language>%
5009             \\bbl@tglobal\<extras\language>%
5010             \\bbl@add\<noextras\language>{%
5011               \XeTeXlinebreaklocale ""}%
5012             \\bbl@tglobal\<noextras\language>}%
5013           \ifx\bbl@ispacesize\undefined
5014             \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
5015           \ifx\AtBeginDocument\@notprerr
5016             \expandafter\@secondoftwo % to execute right now
5017           \fi
5018           \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
5019         \fi}%
5020   \fi}
5021 \ifx\DisableBabelHook\undefined\endinput\fi
5022 \let\bbl@set@renderer\relax
```

```

5023 \let\bbl@unset@renderer\relax
5024 <@Font selection>
5025 \def\bbl@provide@extra#1{}

```

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

```

5026 \def\bbl@xeno-hyph@d{%
5027   \bbl@ifset{\bbl@prehc{\language-name}}%
5028     {\ifnum\hyphenchar\font=\default-hyphenchar
5029       \iffontchar\font\bbl@cl{prehc}\relax
5030       \hyphenchar\font\bbl@cl{prehc}\relax
5031       \else\iffontchar\font"200B
5032         \hyphenchar\font"200B
5033       \else
5034         \bbl@warning
5035           {Neither 0 nor ZERO WIDTH SPACE are available\\%
5036            in the current font, and therefore the hyphen\\%
5037            will be printed. Try changing the fontspec's\\%
5038            'HyphenChar' to another value, but be aware\\%
5039            this setting is not safe (see the manual).\\%
5040            Reported}%
5041         \hyphenchar\font\default-hyphenchar
5042       \fi\fi
5043     \fi}%
5044   {\hyphenchar\font\default-hyphenchar}}

```

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.

```

5045 \ifnum\Xe@alloc@intercharclass<\thr@@
5046   \Xe@alloc@intercharclass\thr@@
5047 \fi
5048 \chardef\bbl@xe@class@default@=\z@
5049 \chardef\bbl@xe@class@cjkideogram@=\@ne
5050 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
5051 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
5052 \chardef\bbl@xe@class@boundary@=4095
5053 \chardef\bbl@xe@class@ignore@=4096

```

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

```

5054 \AddBabelHook{babel-interchar}{beforeextras}{%
5055   \@nameuse{\bbl@xechars{\language-name}}
5056 \DisableBabelHook{babel-interchar}
5057 \protected\def\bbl@charclass#1{%
5058   \ifnum\count@<\z@
5059     \count@-\count@
5060     \loop
5061       \bbl@exp{%
5062         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5063         \XeTeXcharclass\count@ \bbl@tempc
5064         \ifnum\count@<`#1\relax
5065         \advance\count@\@ne
5066       \repeat
5067   \else
5068     \babel@savevariable{\XeTeXcharclass`#1}%
5069     \XeTeXcharclass`#1 \bbl@tempc
5070   \fi
5071   \count@`#1\relax}

```

Now the two user macros. Char classes are declared implicitly, and then the macro to be executed at the `babel-interchar` hook is created. The list of chars to be handled by the hook defined above

has internally the form `\bbl@usingxeclass\bbl@xeclass@punct@english\bbl@charclass{.}`
`\bbl@charclass{,}` (etc.), where `\bbl@usingxeclass` stores the class to be applied to the
subsequent characters. The `\ifcat` part deals with the alternative way to enter characters as macros
(e.g., `\`). As a special case, hyphens are stored as `\bbl@upto`, to deal with ranges.

```

5072 \newcommand\bbl@ifinterchar[1]{%
5073   \let\bbl@tempa\@gobble           % Assume to ignore
5074   \edef\bbl@tempb{\zap@space#1 \@empty}%
5075   \ifx\bbl@KVP@interchar\@nnil\else
5076     \bbl@replace\bbl@KVP@interchar{ }{,}%
5077     \bbl@foreach\bbl@tempb{%
5078       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5079       \ifin@
5080         \let\bbl@tempa\@firstofone
5081       \fi}%
5082   \fi
5083   \bbl@tempa}
5084 \newcommand\IfBabelIntercharT[2]{%
5085   \bbl@carg\bbl@add{\bbl@icsave@\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5086 \newcommand\babelcharclass[3]{%
5087   \EnableBabelHook{babel-interchar}%
5088   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5089   \def\bbl@tempb##1{%
5090     \ifx##1\@empty\else
5091       \ifx##1-%
5092         \bbl@upto
5093       \else
5094         \bbl@charclass{%
5095           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5096       \fi
5097       \expandafter\bbl@tempb
5098     \fi}%
5099   \bbl@ifunset{\bbl@xechars@#1}%
5100   {\toks@{%
5101     \babel@savevariable\XeTeXinterchartokenstate
5102     \XeTeXinterchartokenstate\@ne
5103   }}%
5104   {\toks@\expandafter\expandafter\expandafter{%
5105     \csname bbl@xechars@#1\endcsname}}%
5106   \bbl@csarg\edef{xechars@#1}{%
5107     \the\toks@
5108     \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5109     \bbl@tempb#3\@empty}}
5110 \protected\def\bbl@usingxeclass#1{\count@\zap \let\bbl@tempc#1}
5111 \protected\def\bbl@upto{%
5112   \ifnum\count@>\zap
5113     \advance\count@\@ne
5114     \count@-\count@
5115   \else\ifnum\count@=\zap
5116     \bbl@charclass{-}%
5117   \else
5118     \bbl@error{double-hyphens-class}{\}\}%
5119   \fi\fi}

```

And finally, the command with the code to be inserted. If the language doesn't define a class, then use the global one, as defined above. For the definition there is a intermediate macro, which can be 'disabled' with `\bbl@ic@<label>@<language>`.

```

5120 \def\bbl@ignoreinterchar{%
5121   \ifnum\language=\l@nohyphenation
5122     \expandafter\@gobble
5123   \else
5124     \expandafter\@firstofone
5125   \fi}
5126 \newcommand\babelinterchar[5][]{%

```

```

5127 \let\bbl@kv@label\@empty
5128 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5129 \namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5130 {\bbl@ignoreinterchar{#5}}%
5131 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5132 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5133 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5134 \XeTeXinterchartoks
5135 \@nameuse{bbl@xeclasse@\bbl@tempa @#2}
5136 \bbl@ifunset{bbl@xeclasse@\bbl@tempa @#2}{#2} %
5137 \@nameuse{bbl@xeclasse@\bbl@tempb @#2}
5138 \bbl@ifunset{bbl@xeclasse@\bbl@tempb @#2}{#2} %
5139 = \expandafter%
5140 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5141 \csname\zap@space bbl@xeinter@\bbl@kv@label
5142 @#3@#4@#2 \@empty\endcsname}}}}
5143 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5144 \bbl@ifunset{bbl@ic@#1\@languagename}%
5145 {\bbl@error{unknown-interchar}{#1}{}}%
5146 {\bbl@csarg\let{ic@#1\@languagename}\@firstofone}}
5147 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5148 \bbl@ifunset{bbl@ic@#1\@languagename}%
5149 {\bbl@error{unknown-interchar-b}{#1}{}}%
5150 {\bbl@csarg\let{ic@#1\@languagename}\@gobble}}
5151 </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 T_EX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

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

```

5152 < *xetex | texxet >
5153 \providecommand\bbl@provide@intraspace{}
5154 \bbl@trace{Redefinitions for bidi layout}

Finish here if there is no layout.

5155 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5156 \IfBabelLayout{nopars}
5157 {}
5158 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5159 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5160 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5161 \ifnum\bbl@bidimode>\z@
5162 \IfBabelLayout{pars}
5163 {\def\@hangfrom#1{%
5164 \setbox\@tempboxa\hbox{#1}}%
5165 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5166 \noindent\box\@tempboxa}
5167 \def\raggedright{%
5168 \let\@centercr
5169 \bbl@startskip\z@skip
5170 \@rightskip\@flushglue
5171 \bbl@endskip\@rightskip
5172 \parindent\z@
5173 \parfillskip\bbl@startskip}
5174 \def\raggedleft{%
5175 \let\@centercr
5176 \bbl@startskip\@flushglue
5177 \bbl@endskip\z@skip

```



```

5178 \parindent\z@
5179 \parfillskip\bbl@endskip}}
5180 {}
5181 \fi
5182 \IfBabelLayout{lists}
5183 {\bbl@sreplace\list
5184   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5185   \def\bbl@listleftmargin{%
5186     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5187   \ifcase\bbl@engine
5188     \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5189     \def\p@enumiii{\p@enumii}\theenumii{}\%
5190   \fi
5191   \bbl@sreplace\@verbatim
5192     {\leftskip\@totalleftmargin}%
5193     {\bbl@startskip\textwidth
5194       \advance\bbl@startskip-\linewidth}%
5195   \bbl@sreplace\@verbatim
5196     {\rightskip\z@skip}%
5197     {\bbl@endskip\z@skip}}}%
5198 {}
5199 \IfBabelLayout{contents}
5200 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5201   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5202 {}
5203 \IfBabelLayout{columns}
5204 {\bbl@sreplace\@outputdblcol{\hbxt@\textwidth}{\bbl@outputbox}%
5205   \def\bbl@outputbox#1{%
5206     \hbxt@\textwidth{%
5207       \hskip\columnwidth
5208       \hfil
5209       {\normalcolor\vrule \@width\columnseprule}%
5210       \hfil
5211       \hbxt@\columnwidth{\box\@leftcolumn \hss}%
5212       \hskip-\textwidth
5213       \hbxt@\columnwidth{\box\@outputbox \hss}%
5214       \hskip\columnsep
5215       \hskip\columnwidth}}}%
5216 {}

```

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.

```

5217 \IfBabelLayout{counters*}%
5218 {\bbl@add\bbl@opt@layout{.counters.}%
5219   \AddToHook{shipout/before}{%
5220     \let\bbl@tempa\babelsublr
5221     \let\babelsublr\@firstofone
5222     \let\bbl@save@thepage\thepage
5223     \protected@edef\thepage{\thepage}%
5224     \let\babelsublr\bbl@tempa}%
5225   \AddToHook{shipout/after}{%
5226     \let\thepage\bbl@save@thepage}}{}
5227 \IfBabelLayout{counters}%
5228 {\let\bbl@latinarabic=\@arabic
5229   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5230   \let\bbl@asciroman=\@roman
5231   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5232   \let\bbl@asciiRoman=\@Roman
5233   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5234 \fi % end if layout
5235 </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).

```
5236 < *texxet>
5237 \def\bbl@provide@extra#1{%
5238   % == auto-select encoding ==
5239   \ifx\bbl@encoding@select@off\@empty\else
5240     \bbl@ifunset\bbl@encoding@#1{%
5241       {\def\elt##1{,##1,}%
5242        \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5243        \count@z@
5244        \bbl@foreach\bbl@tempe{%
5245          \def\bbl@tempd{##1}% Save last declared
5246          \advance\count@\@ne}%
5247          \ifnum\count@>\@ne % (1)
5248            \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5249            \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5250            \bbl@replace\bbl@tempa{ }{,}%
5251            \global\bbl@csarg\let{encoding@#1}\@empty
5252            \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5253            \ifin\@else % if main encoding included in ini, do nothing
5254              \let\bbl@tempb\relax
5255              \bbl@foreach\bbl@tempa{%
5256                \ifx\bbl@tempb\relax
5257                  \bbl@xin@{,##1,}{,\bbl@tempe,}%
5258                  \ifin\def\bbl@tempb{##1}\fi
5259                \fi}%
5260              \ifx\bbl@tempb\relax\else
5261                \bbl@exp{%
5262                  \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5263                  \gdef\<bbl@encoding@#1>{%
5264                    \\babel@save\\f@encoding
5265                    \\bbl@add\\originalTeX{\\selectfont}%
5266                    \\fontencoding{\bbl@tempb}%
5267                    \\selectfont}}%
5268                \fi
5269              \fi
5270            \fi}%
5271          }%
5272        \fi}
5273 < /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).

```

5274 (*luatex)
5275 \directlua{ Babel = Babel or {} } % DL2
5276 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5277 \bbl@trace{Read language.dat}
5278 \ifx\bbl@readstream\undefined
5279 \csname newread\endcsname\bbl@readstream
5280 \fi
5281 \begingroup
5282 \toks@{}
5283 \count@\z@ % 0=start, 1=0th, 2=normal
5284 \def\bbl@process@line#1#2 #3 #4 {%
5285 \ifx=#1%
5286 \bbl@process@synonym{#2}%
5287 \else
5288 \bbl@process@language{#1#2}{#3}{#4}%
5289 \fi
5290 \ignorespaces}
5291 \def\bbl@manylang{%
5292 \ifnum\bbl@last>\@ne
5293 \bbl@info{Non-standard hyphenation setup}%
5294 \fi
5295 \let\bbl@manylang\relax}
5296 \def\bbl@process@language#1#2#3{%
5297 \ifcase\count@
5298 \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5299 \or
5300 \count@\tw@
5301 \fi
5302 \ifnum\count@=\tw@
5303 \expandafter\addlanguage\csname l@#1\endcsname
5304 \language\allocationnumber
5305 \chardef\bbl@last\allocationnumber
5306 \bbl@manylang
5307 \let\bbl@elt\relax
5308 \xdef\bbl@languages{%
5309 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5310 \fi
5311 \the\toks@
5312 \toks@{}}
5313 \def\bbl@process@synonym@aux#1#2{%
5314 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5315 \let\bbl@elt\relax
5316 \xdef\bbl@languages{%
5317 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5318 \def\bbl@process@synonym#1{%
5319 \ifcase\count@
5320 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5321 \or

```

```

5322 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5323 \else
5324 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5325 \fi}
5326 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5327 \chardef\l@english\z@
5328 \chardef\l@USenglish\z@
5329 \chardef\bbl@last\z@
5330 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}{}
5331 \gdef\bbl@languages{%
5332 \bbl@elt{english}{0}{hyphen.tex}}{}%
5333 \bbl@elt{USenglish}{0}{}{}}
5334 \else
5335 \global\let\bbl@languages@format\bbl@languages
5336 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5337 \ifnum#2>\z@\else
5338 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5339 \fi}%
5340 \xdef\bbl@languages{\bbl@languages}%
5341 \fi
5342 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}}{} % Define flags
5343 \bbl@languages
5344 \openin\bbl@readstream=language.dat
5345 \ifeof\bbl@readstream
5346 \bbl@warning{I couldn't find language.dat. No additional\\%
5347 patterns loaded. Reported}%
5348 \else
5349 \loop
5350 \endlinechar\m@ne
5351 \read\bbl@readstream to \bbl@line
5352 \endlinechar\^^M
5353 \if T\ifeof\bbl@readstream F\fi T\relax
5354 \ifx\bbl@line\empty\else
5355 \edef\bbl@line{\bbl@line\space\space\space}%
5356 \expandafter\bbl@process@line\bbl@line\relax
5357 \fi
5358 \repeat
5359 \fi
5360 \closein\bbl@readstream
5361 \endgroup
5362 \bbl@trace{Macros for reading patterns files}
5363 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5364 \ifx\babelcatcodetablenum\undefined
5365 \ifx\newcatcodetable\undefined
5366 \def\babelcatcodetablenum{5211}
5367 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5368 \else
5369 \newcatcodetable\babelcatcodetablenum
5370 \newcatcodetable\bbl@pattcodes
5371 \fi
5372 \else
5373 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5374 \fi
5375 \def\bbl@luapatterns#1#2{%
5376 \bbl@get@enc#1::\@@@
5377 \setbox\z@\hbox\bgroup
5378 \begin{group}
5379 \savecatcodetable\babelcatcodetablenum\relax
5380 \initcatcodetable\bbl@pattcodes\relax
5381 \catcodetable\bbl@pattcodes\relax
5382 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5383 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5384 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12

```

```

5385 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5386 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5387 \catcode\`=12 \catcode\'=12 \catcode\"=12
5388 \input #1\relax
5389 \catcodetable\babelcatcodetablenum\relax
5390 \endgroup
5391 \def\bbl@tempa{#2}%
5392 \ifx\bbl@tempa\@empty\else
5393 \input #2\relax
5394 \fi
5395 \egroup}%
5396 \def\bbl@patterns@lua#1{%
5397 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5398 \csname l@#1\endcsname
5399 \edef\bbl@tempa{#1}%
5400 \else
5401 \csname l@#1:\f@encoding\endcsname
5402 \edef\bbl@tempa{#1:\f@encoding}%
5403 \fi\relax
5404 \namedef{lu@texhyphen@loaded@the\language}{}% Temp
5405 \ifundefined{bbl@hyphendata@the\language}%
5406 {\def\bbl@elt##1##2###4{%
5407 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5408 \def\bbl@tempb{##3}%
5409 \ifx\bbl@tempb\@empty\else % if not a synonymous
5410 \def\bbl@tempc{{##3}{##4}}%
5411 \fi
5412 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5413 \fi}%
5414 \bbl@languages
5415 \ifundefined{bbl@hyphendata@the\language}%
5416 {\bbl@info{No hyphenation patterns were set for\%
5417 language '\bbl@tempa'. Reported}}%
5418 {\expandafter\expandafter\expandafter\bbl@luapatterns
5419 \csname bbl@hyphendata@the\language\endcsname}}}%
5420 \endinput\fi

```

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

```

5421 \ifx\DisableBabelHook\@undefined
5422 \AddBabelHook{luatex}{everylanguage}{%
5423 \def\process@language##1##2##3{%
5424 \def\process@line####1####2 ####3 ####4 {}}}
5425 \AddBabelHook{luatex}{loadpatterns}{%
5426 \input #1\relax
5427 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5428 {{#1}}}%
5429 \AddBabelHook{luatex}{loadexceptions}{%
5430 \input #1\relax
5431 \def\bbl@tempb##1##2{{##1}{##1}}%
5432 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5433 {\expandafter\expandafter\expandafter\bbl@tempb
5434 \csname bbl@hyphendata@the\language\endcsname}}
5435 \endinput\fi

```

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

```

5436 \begin{group}
5437 \catcode\%=12
5438 \catcode\'=12
5439 \catcode\"=12
5440 \catcode\:=12
5441 \directlua{
5442 Babel.locale_props = Babel.locale_props or {}
5443 function Babel.lua_error(e, a)

```

```

5444 tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5445 e .. '}' .. (a or '') .. '}{'}')
5446 end
5447
5448 function Babel.bytes(line)
5449     return line:gsub("(.)",
5450         function (chr) return unicode.utf8.char(string.byte(chr)) end)
5451 end
5452
5453 function Babel.priority_in_callback(name,description)
5454     for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5455         if v == description then return i end
5456     end
5457     return false
5458 end
5459
5460 function Babel.begin_process_input()
5461     if luatexbase and luatexbase.add_to_callback then
5462         luatexbase.add_to_callback('process_input_buffer',
5463             Babel.bytes, 'Babel.bytes')
5464     else
5465         Babel.callback = callback.find('process_input_buffer')
5466         callback.register('process_input_buffer', Babel.bytes)
5467     end
5468 end
5469 function Babel.end_process_input ()
5470     if luatexbase and luatexbase.remove_from_callback then
5471         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5472     else
5473         callback.register('process_input_buffer', Babel.callback)
5474     end
5475 end
5476
5477 function Babel.str_to_nodes(fn, matches, base)
5478     local n, head, last
5479     if fn == nil then return nil end
5480     for s in string.utfvalues(fn(matches)) do
5481         if base.id == 7 then
5482             base = base.replace
5483         end
5484         n = node.copy(base)
5485         n.char = s
5486         if not head then
5487             head = n
5488         else
5489             last.next = n
5490         end
5491         last = n
5492     end
5493     return head
5494 end
5495
5496 Babel.linebreaking = Babel.linebreaking or {}
5497 Babel.linebreaking.before = {}
5498 Babel.linebreaking.after = {}
5499 Babel.locale = {}
5500 function Babel.linebreaking.add_before(func, pos)
5501     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5502     if pos == nil then
5503         table.insert(Babel.linebreaking.before, func)
5504     else
5505         table.insert(Babel.linebreaking.before, pos, func)
5506     end

```

```

5507 end
5508 function Babel.linebreaking.add_after(func)
5509     tex.print([[\\noexpand\\csname bbl@luahyphenate\\endcsname]])
5510     table.insert(Babel.linebreaking.after, func)
5511 end
5512
5513 function Babel.addpatterns(pp, lg)
5514     local lg = lang.new(lg)
5515     local pats = lang.patterns(lg) or ''
5516     lang.clear_patterns(lg)
5517     for p in pp:gmatch('[^%s]+') do
5518         ss = ''
5519         for i in string.utfcharacters(p:gsub('%d', '')) do
5520             ss = ss .. '%d?' .. i
5521         end
5522         ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5523         ss = ss:gsub('%.%d%?$', '%%.')
5524         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5525         if n == 0 then
5526             tex.sprint(
5527                 [[\\string\\csname\\space bbl@info\\endcsname{New pattern: }
5528                 .. p .. [{}]])
5529             pats = pats .. ' ' .. p
5530         else
5531             tex.sprint(
5532                 [[\\string\\csname\\space bbl@info\\endcsname{Renew pattern: }
5533                 .. p .. [{}]])
5534         end
5535     end
5536     lang.patterns(lg, pats)
5537 end
5538
5539 Babel.characters = Babel.characters or {}
5540 Babel.ranges = Babel.ranges or {}
5541 function Babel.hlist_has_bidi(head)
5542     local has_bidi = false
5543     local ranges = Babel.ranges
5544     for item in node.traverse(head) do
5545         if item.id == node.id'glyph' then
5546             local itemchar = item.char
5547             local chardata = Babel.characters[itemchar]
5548             local dir = chardata and chardata.d or nil
5549             if not dir then
5550                 for nn, et in ipairs(ranges) do
5551                     if itemchar < et[1] then
5552                         break
5553                     elseif itemchar <= et[2] then
5554                         dir = et[3]
5555                         break
5556                     end
5557                 end
5558             end
5559             if dir and (dir == 'al' or dir == 'r') then
5560                 has_bidi = true
5561             end
5562         end
5563     end
5564     return has_bidi
5565 end
5566 function Babel.set_chranges_b (script, chrng)
5567     if chrng == '' then return end
5568     texio.write('Replacing ' .. script .. ' script ranges')
5569     Babel.script_blocks[script] = {}

```

```

5570     for s, e in string.gmatch(chrng..' ', '(-)%.%(-)%s') do
5571         table.insert(
5572             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5573     end
5574 end
5575
5576 function Babel.discard_sublr(str)
5577     if str:find( [[\string\indexentry]] ) and
5578         str:find( [[\string\babelsublr]] ) then
5579         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5580             function(m) return m:sub(2,-2) end )
5581     end
5582     return str
5583 end
5584 }
5585 \endgroup
5586 \ifx\newattribute\undefined\else % Test for plain
5587     \newattribute\bbl@attr@locale % DL4
5588     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5589     \AddBabelHook{luatex}{beforeextras}{%
5590         \setattribute\bbl@attr@locale\localeid}
5591 \fi
5592 %
5593 \def\BabelStringsDefault{unicode}
5594 \let\luabbl@stop\relax
5595 \AddBabelHook{luatex}{encodedcommands}{%
5596     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5597     \ifx\bbl@tempa\bbl@tempb\else
5598         \directlua{Babel.begin_process_input()}%
5599         \def\luabbl@stop{%
5600             \directlua{Babel.end_process_input()}}%
5601     \fi}%
5602 \AddBabelHook{luatex}{stopcommands}{%
5603     \luabbl@stop
5604     \let\luabbl@stop\relax}
5605 %
5606 \AddBabelHook{luatex}{patterns}{%
5607     \@ifundefined{bbl@hyphendata@the\language}%
5608     {\def\bbl@elt##1##2##3##4{%
5609         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5610         \def\bbl@tempb{##3}%
5611         \ifx\bbl@tempb\@empty\else % if not a synonymous
5612             \def\bbl@tempc{{##3}{##4}}%
5613             \fi
5614             \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5615         \fi}%
5616     \bbl@languages
5617     \@ifundefined{bbl@hyphendata@the\language}%
5618     {\bbl@info{No hyphenation patterns were set for\%
5619         language '#2'. Reported}}%
5620     {\expandafter\expandafter\expandafter\bbl@luapatterns
5621         \csname bbl@hyphendata@the\language\endcsname}}}%
5622 \@ifundefined{bbl@patterns@}{}%
5623 \begin{group}
5624     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
5625     \ifin@ \else
5626         \ifx\bbl@patterns@\@empty \else
5627             \directlua{ Babel.addpatterns(
5628                 [[\bbl@patterns@]], \number\language) }%
5629             \fi
5630             \@ifundefined{bbl@patterns@#1}%
5631             \@empty
5632             {\directlua{ Babel.addpatterns(

```



```

5633          [[\space\csname bbl@patterns@#1\endcsname]],
5634          \number\language) } }%
5635          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5636          \fi
5637        \endgroup}%
5638      \bbl@exp{%
5639        \bbl@ifunset{\bbl@prehc@language}{%
5640          {\bbl@ifblank{\bbl@cs{\prehc@language}}{}}%
5641          {\prehyphenchar=\bbl@cl{\prehc}\relax}}}%

```

\babelpatterns This macro adds patterns. Two macros are used to store them: `\bbl@patterns@` for the global ones and `\bbl@patterns@language` for language ones. We make sure there is a space between words when multiple commands are used.

```

5642 \@onlypreamble\babelpatterns
5643 \AtEndOfPackage{%
5644   \newcommand\babelpatterns[2][\@empty]{%
5645     \ifx\bbl@patterns@relax
5646       \let\bbl@patterns@empty
5647     \fi
5648     \ifx\bbl@pttnlist@empty\else
5649       \bbl@warning{%
5650         You must not intermingle \string\selectlanguage\space and\%
5651         \string\babelpatterns\space or some patterns will not\%
5652         be taken into account. Reported}%
5653       \fi
5654       \ifx@empty#1%
5655         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5656       \else
5657         \edef\bbl@tempb{\zap@space#1 \@empty}%
5658         \bbl@for\bbl@tempa\bbl@tempb{%
5659           \bbl@fixname\bbl@tempa
5660           \bbl@iflanguage\bbl@tempa{%
5661             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5662               \@ifundefined{\bbl@patterns@\bbl@tempa}%
5663               \@empty
5664               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5665               #2}}}%
5666         \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.

```

5667 \def\bbl@intraspace#1 #2 #3\@{
5668   \directlua{
5669     Babel.intraspaces = Babel.intraspaces or {}
5670     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5671       {b = #1, p = #2, m = #3}
5672     Babel.locale_props[\the\localeid].intraspace = %
5673       {b = #1, p = #2, m = #3}
5674   }}
5675 \def\bbl@intrapenalty#1\@{
5676   \directlua{
5677     Babel.intrapenalties = Babel.intrapenalties or {}
5678     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5679     Babel.locale_props[\the\localeid].intrapenalty = #1
5680   }}
5681 \begingroup
5682 \catcode`\%=12
5683 \catcode`\&=14

```

```

5684 \catcode\'=12
5685 \catcode\-=12
5686 \gdef\bbl@seaintraspace{&
5687 \let\bbl@seaintraspace\relax
5688 \directlua{
5689   Babel.sea_enabled = true
5690   Babel.sea_ranges = Babel.sea_ranges or {}
5691   function Babel.set_chranges (script, chrng)
5692     local c = 0
5693     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5694       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5695       c = c + 1
5696     end
5697   end
5698   function Babel.sea_disc_to_space (head)
5699     local sea_ranges = Babel.sea_ranges
5700     local last_char = nil
5701     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5702     for item in node.traverse(head) do
5703       local i = item.id
5704       if i == node.id'glyph' then
5705         last_char = item
5706       elseif i == 7 and item.subtype == 3 and last_char
5707         and last_char.char > 0x0C99 then
5708         quad = font.getfont(last_char.font).size
5709         for lg, rg in pairs(sea_ranges) do
5710           if last_char.char > rg[1] and last_char.char < rg[2] then
5711             lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril1
5712             local intraspace = Babel.intraspaces[lg]
5713             local intrapenalty = Babel.intrapenalties[lg]
5714             local n
5715             if intrapenalty ~= 0 then
5716               n = node.new(14, 0)      &% penalty
5717               n.penalty = intrapenalty
5718               node.insert_before(head, item, n)
5719             end
5720             n = node.new(12, 13)      &% (glue, spaceskip)
5721             node.setglue(n, intraspace.b * quad,
5722               intraspace.p * quad,
5723               intraspace.m * quad)
5724             node.insert_before(head, item, n)
5725             node.remove(head, item)
5726           end
5727         end
5728       end
5729     end
5730   end
5731 }&
5732 \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.

```

5733 \catcode\%=14
5734 \gdef\bbl@cjkintraspaces{&
5735 \let\bbl@cjkintraspaces\relax
5736 \directlua{
5737   require('babel-data-cjk.lua')

```

```

5738 Babel.cjk_enabled = true
5739 function Babel.cjk_linebreak(head)
5740     local GLYPH = node.id'glyph'
5741     local last_char = nil
5742     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5743     local last_class = nil
5744     local last_lang = nil
5745     for item in node.traverse(head) do
5746         if item.id == GLYPH then
5747             local lang = item.lang
5748             local LOCALE = node.get_attribute(item,
5749                 Babel.attr_locale)
5750             local props = Babel.locale_props[LOCALE] or {}
5751             local class = Babel.cjk_class[item.char].c
5752             if props.cjk_quotes and props.cjk_quotes[item.char] then
5753                 class = props.cjk_quotes[item.char]
5754             end
5755             if class == 'cp' then class = 'cl' % ) as CL
5756             elseif class == 'id' then class = 'I'
5757             elseif class == 'cj' then class = 'I' % loose
5758             end
5759             local br = 0
5760             if class and last_class and Babel.cjk_breaks[last_class][class] then
5761                 br = Babel.cjk_breaks[last_class][class]
5762             end
5763             if br == 1 and props.linebreak == 'c' and
5764                 lang ~= \the\l@nohyphenation\space and
5765                 last_lang ~= \the\l@nohyphenation then
5766                 local intrapenalty = props.intrapenalty
5767                 if intrapenalty ~= 0 then
5768                     local n = node.new(14, 0)      % penalty
5769                     n.penalty = intrapenalty
5770                     node.insert_before(head, item, n)
5771                 end
5772                 local intraspace = props.intraspace
5773                 local n = node.new(12, 13)      % (glue, spaceskip)
5774                 node.setglue(n, intraspace.b * quad,
5775                     intraspace.p * quad,
5776                     intraspace.m * quad)
5777                 node.insert_before(head, item, n)
5778             end
5779             if font.getfont(item.font) then
5780                 quad = font.getfont(item.font).size
5781             end
5782             last_class = class
5783             last_lang = lang
5784             else % if penalty, glue or anything else
5785                 last_class = nil
5786             end
5787         end
5788         lang.hyphenate(head)
5789     end
5790 }%
5791 \bbl@luahyphenate}
5792 \gdef\bbl@luahyphenate{%
5793 \let\bbl@luahyphenate\relax
5794 \directlua{
5795     luatexbase.add_to_callback('hyphenate',
5796     function (head, tail)
5797         if Babel.linebreaking.before then
5798             for k, func in ipairs(Babel.linebreaking.before) do
5799                 func(head)
5800             end

```

```

5801     end
5802     lang.hyphenate(head)
5803     if Babel.cjk_enabled then
5804         Babel.cjk_linebreak(head)
5805     end
5806     if Babel.linebreaking.after then
5807         for k, func in ipairs(Babel.linebreaking.after) do
5808             func(head)
5809         end
5810     end
5811     if Babel.set_hboxed then
5812         Babel.set_hboxed(head)
5813     end
5814     if Babel.sea_enabled then
5815         Babel.sea_disc_to_space(head)
5816     end
5817 end,
5818 'Babel.hyphenate')
5819 }}
5820 \endgroup
5821 %
5822 \def\bbl@provide@intraspace{%
5823   \bbl@ifunset\bbl@intsp@\languagename{}\}%
5824   {\expandafter\ifx\cename\bbl@intsp@\languagename\endcsname\@empty\else
5825     \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}}%
5826   \ifin@           % cjk
5827     \bbl@cjk@intraspace
5828     \directlua{
5829       Babel.locale_props = Babel.locale_props or {}
5830       Babel.locale_props[\the\localeid].linebreak = 'c'
5831     }%
5832     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5833     \ifx\bbl@KVP@intrapenalty\@nnil
5834       \bbl@intrapenalty0\@
5835     \fi
5836   \else           % sea
5837     \bbl@sea@intraspace
5838     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\@}%
5839     \directlua{
5840       Babel.sea_ranges = Babel.sea_ranges or {}
5841       Babel.set_chranges('\bbl@cl{sbcpr}',
5842         '\bbl@cl{chrng}')
5843     }%
5844     \ifx\bbl@KVP@intrapenalty\@nnil
5845       \bbl@intrapenalty0\@
5846     \fi
5847   \fi
5848   \fi
5849   \ifx\bbl@KVP@intrapenalty\@nnil\else
5850     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5851   \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`.

```

5852 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5853 \def\bblar@chars{%
5854   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5855   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5856   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5857 \def\bblar@elongated{%
5858   0626,0628,062A,062B,0633,0634,0635,0636,063B,%

```

```

5859 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5860 0649,064A}
5861 \begingroup
5862 \catcode`_ =11 \catcode`: =11
5863 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5864 \endgroup
5865 \gdef\bbl@arabicjust{%
5866 \let\bbl@arabicjust\relax
5867 \newattribute\bblar@kashida
5868 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5869 \bblar@kashida=\z@
5870 \bbl@patchfont{\bbl@parsejalt}}%
5871 \directlua{
5872 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5873 Babel.arabic.elong_map[\the\localeid] = {}
5874 luatexbase.add_to_callback('post_linebreak_filter',
5875 Babel.arabic.justify, 'Babel.arabic.justify')
5876 luatexbase.add_to_callback('hpack_filter',
5877 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5878 }}%

```

Save both node lists to make replacement.

```

5879 \def\bblar@fetchjalt#1#2#3#4{%
5880 \bbl@exp{\bbl@foreach{#1}}{%
5881 \bbl@ifunset\bblar@JE##1}%
5882 {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5883 {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse\bblar@JE##1#2}}%
5884 \directlua{%
5885 local last = nil
5886 for item in node.traverse(tex.box[0].head) do
5887 if item.id == node.id'glyph' and item.char > 0x600 and
5888 not (item.char == 0x200D) then
5889 last = item
5890 end
5891 end
5892 Babel.arabic.#3['##1#4'] = last.char
5893 }}}

```

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?

```

5894 \gdef\bbl@parsejalt{%
5895 \ifx\addfontfeature\undefined\else
5896 \bbl@xin@{e}{\bbl@ccl{lnbrk}}%
5897 \ifin@
5898 \directlua{%
5899 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5900 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5901 tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5902 end
5903 }}%
5904 \fi
5905 \fi}
5906 \gdef\bbl@parsejalti{%
5907 \begingroup
5908 \let\bbl@parsejalt\relax % To avoid infinite loop
5909 \edef\bbl@tempb{\fontid\font}%
5910 \bblar@nofswarn
5911 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5912 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5913 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5914 \addfontfeature{RawFeature+=jalt}%
5915 % \@namedef\bblar@JE0643{06AA}% todo: catch medial kaf
5916 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5917 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%

```

```

5918 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5919 \directlua{%
5920   for k, v in pairs(Babel.arabic.from) do
5921     if Babel.arabic.dest[k] and
5922        not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5923       Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5924       [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5925     end
5926   end
5927 }%
5928 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5929 \begingroup
5930 \catcode`#=11
5931 \catcode`~=11
5932 \directlua{
5933
5934 Babel.arabic = Babel.arabic or {}
5935 Babel.arabic.from = {}
5936 Babel.arabic.dest = {}
5937 Babel.arabic.justify_factor = 0.95
5938 Babel.arabic.justify_enabled = true
5939 Babel.arabic.kashida_limit = -1
5940
5941 function Babel.arabic.justify(head)
5942   if not Babel.arabic.justify_enabled then return head end
5943   for line in node.traverse_id(node.id'hlist', head) do
5944     Babel.arabic.justify_hlist(head, line)
5945   end
5946   % In case the very first item is a line (eg, in \vbox):
5947   while head.prev do head = head.prev end
5948   return head
5949 end
5950
5951 function Babel.arabic.justify_hbox(head, gc, size, pack)
5952   local has_inf = false
5953   if Babel.arabic.justify_enabled and pack == 'exactly' then
5954     for n in node.traverse_id(12, head) do
5955       if n.stretch_order > 0 then has_inf = true end
5956     end
5957     if not has_inf then
5958       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5959     end
5960   end
5961   return head
5962 end
5963
5964 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5965   local d, new
5966   local k_list, k_item, pos_inline
5967   local width, width_new, full, k_curr, wt_pos, goal, shift
5968   local subst_done = false
5969   local elong_map = Babel.arabic.elong_map
5970   local cnt
5971   local last_line
5972   local GLYPH = node.id'glyph'
5973   local KASHIDA = Babel.attr_kashida
5974   local LOCALE = Babel.attr_locale
5975
5976   if line == nil then
5977     line = {}
5978     line.glue_sign = 1

```

```

5979     line.glue_order = 0
5980     line.head = head
5981     line.shift = 0
5982     line.width = size
5983 end
5984
5985 % Exclude last line. todo. But-- it discards one-word lines, too!
5986 % ? Look for glue = 12:15
5987 if (line.glue_sign == 1 and line.glue_order == 0) then
5988     elongs = {}      % Stores elongated candidates of each line
5989     k_list = {}      % And all letters with kashida
5990     pos_inline = 0   % Not yet used
5991
5992     for n in node.traverse_id(GLYPH, line.head) do
5993         pos_inline = pos_inline + 1 % To find where it is. Not used.
5994
5995         % Elongated glyphs
5996         if elong_map then
5997             local locale = node.get_attribute(n, LOCALE)
5998             if elong_map[locale] and elong_map[locale][n.font] and
5999                 elong_map[locale][n.font][n.char] then
6000                 table.insert(elongs, {node = n, locale = locale} )
6001                 node.set_attribute(n.prev, KASHIDA, 0)
6002             end
6003         end
6004
6005         % Tatwil. First create a list of nodes marked with kashida. The
6006         % rest of nodes can be ignored. The list of used weights is build
6007         % when transforms with the key kashida= are declared.
6008         if Babel.kashida_wts then
6009             local k_wt = node.get_attribute(n, KASHIDA)
6010             if k_wt > 0 then % todo. parameter for multi inserts
6011                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
6012             end
6013         end
6014
6015     end % of node.traverse_id
6016
6017     if #elongs == 0 and #k_list == 0 then goto next_line end
6018     full = line.width
6019     shift = line.shift
6020     goal = full * Babel.arabic.justify_factor % A bit crude
6021     width = node.dimensions(line.head) % The 'natural' width
6022
6023     % == Elongated ==
6024     % Original idea taken from 'chickenize'
6025     while (#elongs > 0 and width < goal) do
6026         subst_done = true
6027         local x = #elongs
6028         local curr = elongs[x].node
6029         local oldchar = curr.char
6030         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6031         width = node.dimensions(line.head) % Check if the line is too wide
6032         % Substitute back if the line would be too wide and break:
6033         if width > goal then
6034             curr.char = oldchar
6035             break
6036         end
6037         % If continue, pop the just substituted node from the list:
6038         table.remove(elongs, x)
6039     end
6040
6041     % == Tatwil ==

```

```

6042 % Traverse the kashida node list so many times as required, until
6043 % the line is filled. The first pass adds a tatweel after each
6044 % node with kashida in the line, the second pass adds another one,
6045 % and so on. In each pass, add first the kashida with the highest
6046 % weight, then with lower weight and so on.
6047 if #k_list == 0 then goto next_line end
6048
6049 width = node.dimensions(line.head) % The 'natural' width
6050 k_curr = #k_list % Traverse backwards, from the end
6051 wt_pos = 1
6052
6053 while width < goal do
6054     subst_done = true
6055     k_item = k_list[k_curr].node
6056     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6057         d = node.copy(k_item)
6058         d.char = 0x0640
6059         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6060         d.xoffset = 0
6061         line.head, new = node.insert_after(line.head, k_item, d)
6062         width_new = node.dimensions(line.head)
6063         if width > goal or width == width_new then
6064             node.remove(line.head, new) % Better compute before
6065             break
6066         end
6067         if Babel.fix_diacr then
6068             Babel.fix_diacr(k_item.next)
6069         end
6070         width = width_new
6071     end
6072     if k_curr == 1 then
6073         k_curr = #k_list
6074         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6075     else
6076         k_curr = k_curr - 1
6077     end
6078 end
6079
6080 % Limit the number of tatweel by removing them. Not very efficient,
6081 % but it does the job in a quite predictable way.
6082 if Babel.arabic.kashida_limit > -1 then
6083     cnt = 0
6084     for n in node.traverse_id(GLYPH, line.head) do
6085         if n.char == 0x0640 then
6086             cnt = cnt + 1
6087             if cnt > Babel.arabic.kashida_limit then
6088                 node.remove(line.head, n)
6089             end
6090         else
6091             cnt = 0
6092         end
6093     end
6094 end
6095
6096 ::next_line::
6097
6098 % Must take into account marks and ins, see luatex manual.
6099 % Have to be executed only if there are changes. Investigate
6100 % what's going on exactly.
6101 if subst_done and not gc then
6102     d = node.hpack(line.head, full, 'exactly')
6103     d.shift = shift
6104     node.insert_before(head, line, d)

```



```

6105      node.remove(head, line)
6106    end
6107  end % if process line
6108 end
6109 }
6110 \endgroup
6111 \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`.

```

6112 \def\bbl@scr@node@list{%
6113   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6114   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6115 \ifnum\bbl@bidimode=102 % bidi-r
6116   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6117 \fi
6118 \def\bbl@set@renderer{%
6119   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6120   \ifin@
6121     \let\bbl@unset@renderer\relax
6122   \else
6123     \bbl@exp{%
6124       \def\\bbl@unset@renderer{%
6125         \def<g__fontspec_default_fontopts_clist>{%
6126           \[g__fontspec_default_fontopts_clist]}}%
6127       \def<g__fontspec_default_fontopts_clist>{%
6128         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
6129     \fi}
6130 <@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.

```

6131 \directlua{% DL6
6132 Babel.script_blocks = {
6133   ['dflt'] = {},
6134   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6135               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6136   ['Armn'] = {{0x0530, 0x058F}},
6137   ['Beng'] = {{0x0980, 0x09FF}},
6138   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0ABBF}},
6139   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6140   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6141              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6142   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6143   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6144              {0xAB00, 0xAB2F}},
6145   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6146   % Don't follow strictly Unicode, which places some Coptic letters in
6147   % the 'Greek and Coptic' block

```

```

6148 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6149 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6150             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6151             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6152             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6153             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6154             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6155 ['Hebr'] = {{0x0590, 0x05FF},
6156             {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6157 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6158             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6159 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6160 ['Knda'] = {{0x0C80, 0x0CFF}},
6161 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6162             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6163             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6164 ['Lao'] = {{0x0E80, 0x0EFF}},
6165 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6166             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6167             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6168 ['Mahj'] = {{0x11150, 0x1117F}},
6169 ['Mlym'] = {{0x0D00, 0x0D7F}},
6170 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6171 ['Orya'] = {{0x0B00, 0x0B7F}},
6172 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6173 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6174 ['Taml'] = {{0x0B80, 0x0BFF}},
6175 ['Telu'] = {{0x0C00, 0x0C7F}},
6176 ['Tfng'] = {{0x2D30, 0x2D7F}},
6177 ['Thai'] = {{0x0E00, 0x0E7F}},
6178 ['Tibt'] = {{0x0F00, 0x0FFF}},
6179 ['Vaii'] = {{0xA500, 0xA63F}},
6180 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6181 }
6182
6183 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6184 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6185 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6186
6187 function Babel.locale_map(head)
6188   if not Babel.locale_mapped then return head end
6189
6190   local LOCALE = Babel.attr_locale
6191   local GLYPH = node.id('glyph')
6192   local inmath = false
6193   local toloc_save
6194   for item in node.traverse(head) do
6195     local toloc
6196     if not inmath and item.id == GLYPH then
6197       % Optimization: build a table with the chars found
6198       if Babel.chr_to_loc[item.char] then
6199         toloc = Babel.chr_to_loc[item.char]
6200       else
6201         for lc, maps in pairs(Babel.loc_to_scr) do
6202           for _, rg in pairs(maps) do
6203             if item.char >= rg[1] and item.char <= rg[2] then
6204               Babel.chr_to_loc[item.char] = lc
6205               toloc = lc
6206             end
6207           end
6208         end
6209       end
6210       % Treat composite chars in a different fashion, because they

```

```

6211      % 'inherit' the previous locale.
6212      if (item.char >= 0x0300 and item.char <= 0x036F) or
6213         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6214         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6215         Babel.chr_to_loc[item.char] = -2000
6216         toloc = -2000
6217     end
6218     if not toloc then
6219         Babel.chr_to_loc[item.char] = -1000
6220     end
6221 end
6222 if toloc == -2000 then
6223     toloc = toloc_save
6224 elseif toloc == -1000 then
6225     toloc = nil
6226 end
6227 if toloc and Babel.locale_props[toloc] and
6228     Babel.locale_props[toloc].letters and
6229     tex.getcatcode(item.char) \string~= 11 then
6230     toloc = nil
6231 end
6232 if toloc and Babel.locale_props[toloc].script
6233     and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6234     and Babel.locale_props[toloc].script ==
6235     Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6236     toloc = nil
6237 end
6238 if toloc then
6239     if Babel.locale_props[toloc].lg then
6240         item.lang = Babel.locale_props[toloc].lg
6241         node.set_attribute(item, LOCALE, toloc)
6242     end
6243     if Babel.locale_props[toloc]['/'..item.font] then
6244         item.font = Babel.locale_props[toloc]['/'..item.font]
6245     end
6246 end
6247 toloc_save = toloc
6248 elseif not inmath and item.id == 7 then % Apply recursively
6249     item.replace = item.replace and Babel.locale_map(item.replace)
6250     item.pre      = item.pre and Babel.locale_map(item.pre)
6251     item.post     = item.post and Babel.locale_map(item.post)
6252 elseif item.id == node.id'math' then
6253     inmath = (item.subtype == 0)
6254 end
6255 end
6256 return head
6257 end
6258 }

```

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

```

6259 \newcommand\babelcharproperty[1]{%
6260   \count@=#1\relax
6261   \ifvmode
6262     \expandafter\babel@chprop
6263   \else
6264     \babel@error{charproperty-only-vertical}{#1}{#1}%
6265   \fi}
6266 \newcommand\babel@chprop[3][\the\count@]{%
6267   \@tempcnta=#1\relax
6268   \babel@ifunset{babel@chprop@#2}% {unknown-char-property}
6269   {\babel@error{unknown-char-property}{#2}{#2}}%
6270   }%

```

```

6271 \loop
6272   \bbl@cs{chprop@#2}{#3}%
6273   \ifnum\count@<\@tempcnta
6274     \advance\count@\@ne
6275   \repeat}
6276 %
6277 \def\bbl@chprop@direction#1{%
6278   \directlua{
6279     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6280     Babel.characters[\the\count@]['d'] = '#1'
6281   }}
6282 \let\bbl@chprop@bc\bbl@chprop@direction
6283 %
6284 \def\bbl@chprop@mirror#1{%
6285   \directlua{
6286     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6287     Babel.characters[\the\count@]['m'] = '\number#1'
6288   }}
6289 \let\bbl@chprop@bmg\bbl@chprop@mirror
6290 %
6291 \def\bbl@chprop@linebreak#1{%
6292   \directlua{
6293     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6294     Babel.cjk_characters[\the\count@]['c'] = '#1'
6295   }}
6296 \let\bbl@chprop@lb\bbl@chprop@linebreak
6297 %
6298 \def\bbl@chprop@locale#1{%
6299   \directlua{
6300     Babel.chr_to_loc = Babel.chr_to_loc or {}
6301     Babel.chr_to_loc[\the\count@] =
6302       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
6303   }}

```

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.

```

6304 \directlua{% DL7
6305   Babel.nohyphenation = \the\l@nohyphenation
6306 }

```

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).

```

6307 \begingroup
6308 \catcode`\~ = 12
6309 \catcode`\% = 12
6310 \catcode`\& = 14
6311 \catcode`\| = 12
6312 \gdef\babelprehyphenation{%&
6313   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}[{}]}
6314 \gdef\babelposthyphenation{%&
6315   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}[{}]}
6316 %
6317 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6318   \ifcase#1
6319     \bbl@activateprehyphen
6320   \or
6321     \bbl@activateposthyphen

```

```

6322 \fi
6323 \beginingroup
6324 \def\babeltempa{\bbl@add@list\babeltempb}&%
6325 \let\babeltempb\@empty
6326 \def\bbl@tempa{#5}&%
6327 \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
6328 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6329 \bbl@ifsamestring{##1}{remove}&%
6330 {\bbl@add@list\babeltempb{nil}}&%
6331 {\directlua{
6332     local rep = [=##1]=]
6333     local three_args = '%s*=%s*([%-d%.%a{|}|+)%s+([%-d%.%a{|}|+)%s+([%-d%.%a{|}|+)'
6334     &% Numeric passes directly: kern, penalty...
6335     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6336     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6337     rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6338     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6339     rep = rep:gsub('node%s*=%s*([a+])%s*([a+])', Babel.capture_node)
6340     rep = rep:gsub(' (norule)' .. three_args,
6341         'norule = {' .. '%2, %3, %4' .. '}')
6342     if #1 == 0 or #1 == 2 then
6343         rep = rep:gsub(' (space)' .. three_args,
6344             'space = {' .. '%2, %3, %4' .. '}')
6345         rep = rep:gsub(' (spacefactor)' .. three_args,
6346             'spacefactor = {' .. '%2, %3, %4' .. '}')
6347         rep = rep:gsub(' (kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6348         &% Transform values
6349         rep, n = rep:gsub(' {([%a%-%.]+)|([%a%_%.]+)}',
6350             function(v,d)
6351                 return string.format (
6352                     '{\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6353                     v,
6354                     load( 'return Babel.locale_props'..
6355                         '[\the\csname bbl@id@@#3\endcsname].' .. d)() )
6356                 end )
6357         rep, n = rep:gsub(' {([%a%-%.]+)|([%-d%.]+)}',
6358             '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6359     end
6360     if #1 == 1 then
6361         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6362         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6363         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6364     end
6365     tex.print([\string\babeltempa{[]] .. rep .. [[]]])
6366 }}&%
6367 \bbl@foreach\babeltempb{&%
6368 \bbl@forkv{##1}{&%
6369 \in@{,###1},{,nil,step,data,remove,insert,string,no,pre,no,&%
6370 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6371 \ifin@else
6372 \bbl@error{bad-transform-option}{###1}{,}{&%
6373 \fi}}&%
6374 \let\bbl@kv@attribute\relax
6375 \let\bbl@kv@label\relax
6376 \let\bbl@kv@fonts\@empty
6377 \let\bbl@kv@prepend\relax
6378 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
6379 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6380 \ifx\bbl@kv@attribute\relax
6381 \ifx\bbl@kv@label\relax\else
6382 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6383 \bbl@replace\bbl@kv@fonts{ ,}&%
6384 \edef\bbl@kv@attribute{\bbl@ATR@{\bbl@kv@label @#3\bbl@kv@fonts}}&%

```

```

6385 \count@z@
6386 \def\bbl@elt##1##2##3{%&
6387 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6388 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6389 {\count@z@ne}&%
6390 {\bbl@error{font-conflict-transforms}{}}{}}}&%
6391 {}}&%
6392 \bbl@transfont@list
6393 \ifnum\count@=z@
6394 \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6395 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6396 \fi
6397 \bbl@ifunset{\bbl@kv@attribute}&%
6398 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6399 {}&%
6400 \global\bbl@carg\setattribute{\bbl@kv@attribute}@ne
6401 \fi
6402 \else
6403 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6404 \fi
6405 \directlua{
6406 local lbkr = Babel.linebreaking.replacements[#1]
6407 local u = unicode.utf8
6408 local id, attr, label
6409 if #1 == 0 then
6410 id = \the\csname bbl@id@@#3\endcsname\space
6411 else
6412 id = \the\csname l@#3\endcsname\space
6413 end
6414 \ifx\bbl@kv@attribute\relax
6415 attr = -1
6416 \else
6417 attr = luatexbase.registernumber'\bbl@kv@attribute'
6418 \fi
6419 \ifx\bbl@kv@label\relax\else &% Same refs:
6420 label = [==[\bbl@kv@label]==]
6421 \fi
6422 &% Convert pattern:
6423 local patt = string.gsub([==[#4]==], '%s', '')
6424 if #1 == 0 then
6425 patt = string.gsub(patt, '|', ' ')
6426 end
6427 if not u.find(patt, '()', nil, true) then
6428 patt = '()' .. patt .. '()'
6429 end
6430 if #1 == 1 then
6431 patt = string.gsub(patt, '%(%)%^', '^()')
6432 patt = string.gsub(patt, '%$(%)', '()$')
6433 end
6434 patt = u.gsub(patt, '{(.)}',
6435 function (n)
6436 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6437 end)
6438 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6439 function (n)
6440 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6441 end)
6442 lbkr[id] = lbkr[id] or {}
6443 table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6444 { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6445 }&%
6446 \endgroup}
6447 \endgroup

```

```

6448 %
6449 \let\bbl@transfont@list\@empty
6450 \def\bbl@settransfont{%
6451   \global\let\bbl@settransfont\relax % Execute only once
6452   \gdef\bbl@transfont{%
6453     \def\bbl@elt####1####2####3{%
6454       \bbl@ifblank{####3}%
6455       {\count@tw@}% Do nothing if no fonts
6456       {\count@z@
6457         \bbl@vforeach{####3}{%
6458           \def\bbl@tempd{#####1}%
6459           \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6460           \ifx\bbl@tempd\bbl@tempe
6461             \count@ne
6462           \else\ifx\bbl@tempd\bbl@transfam
6463             \count@ne
6464           \fi\fi}%
6465           \ifcase\count@
6466             \bbl@csarg\unsetattribute{ATR####2@####1@####3}%
6467           \or
6468             \bbl@csarg\setattribute{ATR####2@####1@####3}\@ne
6469           \fi}}%
6470     \bbl@transfont@list}%
6471 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6472 \gdef\bbl@transfam{-unknown-}%
6473 \bbl@foreach\bbl@font@fams{%
6474   \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6475   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6476   {\xdef\bbl@transfam{##1}}%
6477   {}}}
6478 %
6479 \DeclareRobustCommand\enablelocaletransform[1]{%
6480   \bbl@ifunset\bbl@ATR#1@\language @}%
6481   {\bbl@error{transform-not-available}{#1}{}}%
6482   {\bbl@csarg\setattribute{ATR#1@\language @}\@ne}}
6483 \DeclareRobustCommand\disablelocaletransform[1]{%
6484   \bbl@ifunset\bbl@ATR#1@\language @}%
6485   {\bbl@error{transform-not-available-b}{#1}{}}%
6486   {\bbl@csarg\unsetattribute{ATR#1@\language @}}}

```

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

```

6487 \def\bbl@activateposthyphen{%
6488   \let\bbl@activateposthyphen\relax
6489   \ifx\bbl@attr@hboxed\undefined
6490     \newattribute\bbl@attr@hboxed
6491   \fi
6492   \directlua{
6493     require('babel-transforms.lua')
6494     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6495   }}
6496 \def\bbl@activateprehyphen{%
6497   \let\bbl@activateprehyphen\relax
6498   \ifx\bbl@attr@hboxed\undefined
6499     \newattribute\bbl@attr@hboxed
6500   \fi
6501   \directlua{
6502     require('babel-transforms.lua')
6503     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6504   }}
6505 \newcommand\SetTransformValue[3]{%
6506   \directlua{
6507     Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3

```

```
6508  }}
```

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.

```
6509 \newcommand\ShowBabelTransforms[1]{%
6510   \bbl@activateprehyphen
6511   \bbl@activateposthyphen
6512   \begingroup
6513     \directlua{ Babel.show_transforms = true }%
6514     \setbox\z@\vbox{#1}%
6515     \directlua{ Babel.show_transforms = false }%
6516   \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.

```
6517 \newcommand\localeprehyphenation[1]{%
6518   \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.

```
6519 \def\bbl@activate@preotf{%
6520   \let\bbl@activate@preotf\relax % only once
6521   \directlua{
6522     function Babel.pre_otfload_v(head)
6523       if Babel.numbers and Babel.digits_mapped then
6524         head = Babel.numbers(head)
6525       end
6526       if Babel.bidi_enabled then
6527         head = Babel.bidi(head, false, dir)
6528       end
6529       return head
6530     end
6531     %
6532     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6533       if Babel.numbers and Babel.digits_mapped then
6534         head = Babel.numbers(head)
6535       end
6536       if Babel.bidi_enabled then
6537         head = Babel.bidi(head, false, dir)
6538       end
6539       return head
6540     end
6541     %
6542     luatexbase.add_to_callback('pre_linebreak_filter',
6543       Babel.pre_otfload_v,
6544       'Babel.pre_otfload_v',
6545       Babel.priority_in_callback('pre_linebreak_filter',
6546         'luaotfload.node_processor') or nil)
6547     %
6548     luatexbase.add_to_callback('hpack_filter',
6549       Babel.pre_otfload_h,
6550       'Babel.pre_otfload_h',
6551       Babel.priority_in_callback('hpack_filter',
6552         'luaotfload.node_processor') or nil)
6553   }}
```


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`.

```

6554 \breakafterdirmode=1
6555 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6556 \let\bbl@beforeforeign\leavevmode
6557 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6558 \RequirePackage{luatexbase}
6559 \bbl@activate@preotf
6560 \directlua{
6561   require('babel-data-bidi.lua')
6562   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6563     require('babel-bidi-basic.lua')
6564   \or
6565     require('babel-bidi-basic-r.lua')
6566     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6567     table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6568     table.insert(Babel.ranges, {0x10000, 0x10FFFD, 'on'})
6569   \fi}
6570 \newattribute\bbl@attr@dir
6571 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6572 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6573 \fi
6574 %
6575 \chardef\bbl@thetextdir\z@
6576 \chardef\bbl@thepardir\z@
6577 \def\bbl@getluadir#1{%
6578   \directlua{
6579     if tex.#1dir == 'TLT' then
6580       tex.sprint('0')
6581     elseif tex.#1dir == 'TRT' then
6582       tex.sprint('1')
6583     else
6584       tex.sprint('0')
6585     end}}
6586 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6587   \ifcase#3\relax
6588     \ifcase\bbl@getluadir{#1}\relax\else
6589       #2 TLT\relax
6590     \fi
6591   \else
6592     \ifcase\bbl@getluadir{#1}\relax
6593       #2 TRT\relax
6594     \fi
6595   \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).

6596 \def\bbl@thedir{0}
6597 \def\bbl@textdir#1{%
6598   \bbl@setluadir{text}\textdir{#1}%
6599   \chardef\bbl@thetextdir#1\relax
6600   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6601   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6602 \def\bbl@pardir#1{% Used twice
6603   \bbl@setluadir{par}\pardir{#1}%
6604   \chardef\bbl@thepardir#1\relax}
6605 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6606 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6607 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6608 \ifnum\bbl@bidimode>\z@ % Any bidi=
6609 \def\bbl@insidemath{0}%
6610 \def\bbl@everymath{\def\bbl@insidemath{1}}
6611 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6612 \frozen@everymath\expandafter{%
6613   \expandafter\bbl@everymath\the\frozen@everymath}
6614 \frozen@everydisplay\expandafter{%
6615   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6616 \AtBeginDocument{
6617   \directlua{
6618     function Babel.math_box_dir(head)
6619       if not (token.get_macro('bbl@insidemath') == '0') then
6620         if Babel.hlist_has_bidi(head) then
6621           local d = node.new(node.id'dir')
6622           d.dir = '+TRT'
6623           node.insert_before(head, node.has_glyph(head), d)
6624           local inmath = false
6625           for item in node.traverse(head) do
6626             if item.id == 11 then
6627               inmath = (item.subtype == 0)
6628             elseif not inmath then
6629               node.set_attribute(item,
6630                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6631             end
6632           end
6633         end
6634       end
6635       return head
6636     end
6637     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6638       "Babel.math_box_dir", 0)
6639     if Babel.unset_atdir then
6640       luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6641         "Babel.unset_atdir")
6642       luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6643         "Babel.unset_atdir")
6644     end
6645   }}%
6646 \fi

Experimental. Tentative name.

6647 \DeclareRobustCommand\localebox[1]{%
6648   {\def\bbl@insidemath{0}%
6649     \mbox{\foreignlanguage{\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 `bidi=basic`, without having to patch almost any macro where text direction is relevant.

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

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

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

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

```

6650 \bbl@trace{Redefinitions for bidi layout}
6651 %
6652 <<{*More package options}>> ≡
6653 \chardef\bbl@eqnpos\z@
6654 \DeclareOption{leqno}{\chardef\bbl@eqnpos@ne}
6655 \DeclareOption{fleqn}{\chardef\bbl@eqnpos@tw@}
6656 <</More package options>>
6657 %
6658 \ifnum\bbl@bidimode>\z@ % Any bidi=
6659   \matheqdirmode@ne % A luatex primitive
6660   \mathemptydisplaymode@ne % Another
6661   \let\bbl@eqnodir\relax
6662   \def\bbl@eqdel{()}
6663   \def\bbl@eqnum{%
6664     {\normalfont\normalcolor
6665       \expandafter\@firstoftwo\bbl@eqdel
6666       \theequation
6667       \expandafter\@secondoftwo\bbl@eqdel}}
6668   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6669   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6670   \def\bbl@eqno@flip#1{%
6671     \ifdim\predisplaysize=-\maxdimen
6672       \eqno
6673       \hb@xt@.01pt{%
6674         \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6675     \else
6676       \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6677     \fi
6678     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6679   \def\bbl@leqno@flip#1{%
6680     \ifdim\predisplaysize=-\maxdimen
6681       \leqno
6682       \hb@xt@.01pt{%
6683         \hss\hb@xt@\displaywidth{#1}\glet\bbl@upset\@currentlabel}\hss}%
6684     \else
6685       \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6686     \fi
6687     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6688 %
6689 \AtBeginDocument{%
6690   \ifx\bbl@noamsmath\relax\else
6691     \ifx\maketag@@@ \@undefined % Normal equation, eqnarray
6692       \AddToHook{env/equation/begin}{%
6693         \ifnum\bbl@thetextdir>\z@
6694           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6695           \let\@eqnnum\bbl@eqnum
6696           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6697           \chardef\bbl@thetextdir\z@
6698           \bbl@add\normalfont{\bbl@eqnodir}%
6699           \ifcase\bbl@eqnpos
6700             \let\bbl@puteqno\bbl@eqno@flip
6701           \or
6702             \let\bbl@puteqno\bbl@leqno@flip
6703           \fi
6704           \fi}%
6705       \ifnum\bbl@eqnpos=\tw@\else
6706         \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6707       \fi
6708       \AddToHook{env/eqnarray/begin}{%

```

```

6709 \ifnum\bbbl@thetextdir>\z@
6710 \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6711 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6712 \chardef\bbbl@thetextdir\z@
6713 \bbbl@add\normalfont{\bbbl@eqnodir}%
6714 \ifnum\bbbl@eqnpos=\@ne
6715 \def\@eqnnum{%
6716 \setbox\z@\hbox{\bbbl@eqnum}%
6717 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6718 \else
6719 \let\@eqnnum\bbbl@eqnum
6720 \fi
6721 \fi}
6722 % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6723 \expandafter\bbbl@sreplace\csname \endcsname{${\eqno\kern.001pt$}}%
6724 \expandafter\bbbl@sreplace\csname \endcsname
6725 {\dollar\dollar@end}{\eqno\kern.001pt\dollar\dollar@end}%
6726 \else % amstex
6727 \bbbl@exp{% Hack to hide maybe undefined conditionals:
6728 \chardef\bbbl@eqnpos=0%
6729 \<iftagsleft>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6730 \ifnum\bbbl@eqnpos=\@ne
6731 \let\bbbl@ams@lap\hbox
6732 \else
6733 \let\bbbl@ams@lap\llap
6734 \fi
6735 \ExplSyntaxOn % Required by \bbbl@sreplace with \intertext@
6736 \bbbl@sreplace\intertext@{\normalbaselines}%
6737 {\normalbaselines
6738 \ifx\bbbl@eqnodir\relax\else\bbbl@paddir\@ne\bbbl@eqnodir\fi}%
6739 \ExplSyntaxOff
6740 \def\bbbl@ams@tagbox#1#2{#1{\bbbl@eqnodir#2}}% #1=hbox|@lap|flip
6741 \ifx\bbbl@ams@lap\hbox % leqno
6742 \def\bbbl@ams@flip#1{%
6743 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6744 \else % eqno
6745 \def\bbbl@ams@flip#1{%
6746 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}%
6747 \fi
6748 \def\bbbl@ams@preset#1{%
6749 \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6750 \ifnum\bbbl@thetextdir>\z@
6751 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6752 \bbbl@sreplace\textdef@{\hbox}{\bbbl@ams@tagbox\hbox}%
6753 \bbbl@sreplace\maketag@@@{\hbox}{\bbbl@ams@tagbox#1}%
6754 \fi}%
6755 \ifnum\bbbl@eqnpos=\tw@ \else
6756 \def\bbbl@ams@equation{%
6757 \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6758 \ifnum\bbbl@thetextdir>\z@
6759 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6760 \chardef\bbbl@thetextdir\z@
6761 \bbbl@add\normalfont{\bbbl@eqnodir}%
6762 \ifcase\bbbl@eqnpos
6763 \def\veqno##1##2{\bbbl@eqno@flip{##1##2}}%
6764 \or
6765 \def\veqno##1##2{\bbbl@leqno@flip{##1##2}}%
6766 \fi
6767 \fi}%
6768 \AddToHook{env/equation/begin}{\bbbl@ams@equation}%
6769 \AddToHook{env/equation*/begin}{\bbbl@ams@equation}%
6770 \fi
6771 \AddToHook{env/cases/begin}{\bbbl@ams@preset\bbbl@ams@lap}%

```

```

6772 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6773 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6774 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6775 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6776 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6777 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6778 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6779 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6780 % Hackish, for proper alignment. Don't ask me why it works!:
6781 \bbl@exp{% Avoid a 'visible' conditional
6782   \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}\<fi>}%
6783   \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{}\<fi>}}%
6784 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6785 \AddToHook{env/split/before}{%
6786   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6787   \ifnum\bbl@thetextdir>\z@
6788     \bbl@ifsamestring\@currentvir{equation}%
6789     {\ifx\bbl@ams@lap\hbox % legno
6790       \def\bbl@ams@flip#1{%
6791         \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6792       \else
6793         \def\bbl@ams@flip#1{%
6794           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6795       \fi}%
6796     }%
6797   \fi}%
6798 \fi\fi}
6799 \fi

```

Declarations specific to lua, called by \babelprovide.

```

6800 \def\bbl@provide@extra#1{%
6801   % == onchar ==
6802   \ifx\bbl@KVP@onchar\@nnil\else
6803     \bbl@luahyphenate
6804     \bbl@exp{%
6805       \\\AddToHook{env/document/before}{%
6806         {\let\\bbl@ifrestoring\\@firstoftwo
6807           \\\select@language{#1}{}}}%
6808       \directlua{
6809         if Babel.locale_mapped == nil then
6810           Babel.locale_mapped = true
6811           Babel.linebreaking.add_before(Babel.locale_map, 1)
6812           Babel.loc_to_scr = {}
6813           Babel.chr_to_loc = Babel.chr_to_loc or {}
6814         end
6815         Babel.locale_props[\the\localeid].letters = false
6816       }%
6817       \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6818       \ifin@
6819         \directlua{
6820           Babel.locale_props[\the\localeid].letters = true
6821         }%
6822       \fi
6823       \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6824       \ifin@
6825         \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6826           \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6827         \fi
6828         \bbl@exp{\\bbl@add\\bbl@starthyphens
6829           {\bbl@patterns@lua{\language\name}}}%
6830         \directlua{
6831           if Babel.script_blocks['\bbl@cl{sbc}'] then
6832             Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']

```

```

6833         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l\@language}\space
6834     end
6835 }%
6836 \fi
6837 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6838 \ifin@
6839     \bbl@ifunset{bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
6840     \bbl@ifunset{bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
6841     \directlua{
6842         if Babel.script_blocks['\bbl@cl{sbc}'] then
6843             Babel.loc_to_scr[\the\localeid] =
6844                 Babel.script_blocks['\bbl@cl{sbc}']
6845         end}%
6846     \ifx\bbl@mapselect\@undefined
6847         \AtBeginDocument{%
6848             \bbl@patchfont{\bbl@mapselect}}%
6849             {\selectfont}}%
6850     \def\bbl@mapselect{%
6851         \let\bbl@mapselect\relax
6852         \edef\bbl@prefontid{\fontid\font}}%
6853     \def\bbl@mapdir##1{%
6854         \begingroup
6855             \setbox\z@\hbox{% Force text mode
6856                 \def\language{##1}%
6857                 \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6858                 \bbl@switchfont
6859                 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6860                     \directlua{
6861                         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6862                             ['\bbl@prefontid'] = \fontid\font\space}%
6863                     }%
6864                 \endgroup}%
6865     \fi
6866     \bbl@exp{\bbl@add{\bbl@mapselect{\bbl@mapdir{\language}}}}%
6867 \fi
6868 \fi
6869 % == mapfont ==
6870 % For bidi texts, to switch the font based on direction. Deprecated
6871 \ifx\bbl@KVP@mapfont\@nnil\else
6872     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
6873     {\bbl@error{unknown-mapfont}}{}{}%
6874     \bbl@ifunset{bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
6875     \bbl@ifunset{bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
6876     \ifx\bbl@mapselect\@undefined
6877         \AtBeginDocument{%
6878             \bbl@patchfont{\bbl@mapselect}}%
6879             {\selectfont}}%
6880     \def\bbl@mapselect{%
6881         \let\bbl@mapselect\relax
6882         \edef\bbl@prefontid{\fontid\font}}%
6883     \def\bbl@mapdir##1{%
6884         {\def\language{##1}%
6885         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6886         \bbl@switchfont
6887         \directlua{Babel.fontmap
6888             [\the\csname bbl@wdir@##1\endcsname]%
6889             [\bbl@prefontid]=\fontid\font}}}%
6890     \fi
6891     \bbl@exp{\bbl@add{\bbl@mapselect{\bbl@mapdir{\language}}}}%
6892 \fi
6893 % == Line breaking: CJK quotes ==
6894 \ifcase\bbl@engine\or
6895     \bbl@xin@{/c}{\bbl@cl{\lnbrk}}%

```

```

6896 \ifin@
6897 \bbl@ifunset{bbl@quote@\languagename}{}%
6898 {\directlua{
6899     Babel.locale_props[\the\localeid].CJK_quotes = {}
6900     local cs = 'op'
6901     for c in string.utfvalues(
6902         [[\csname bbl@quote@\languagename\endcsname]]) do
6903         if Babel.cjk_characters[c].c == 'qu' then
6904             Babel.locale_props[\the\localeid].CJK_quotes[c] = cs
6905         end
6906         cs = (cs == 'op') and 'cl' or 'op'
6907     end
6908 }}%
6909 \fi
6910 \fi
6911 % == Counters: mapdigits ==
6912 % Native digits
6913 \ifx\bbl@KVP@mapdigits\@nnil\else
6914 \bbl@ifunset{bbl@dgnat@\languagename}{}%
6915 {\bbl@activate@preotf
6916 \directlua{
6917     Babel.digits_mapped = true
6918     Babel.digits = Babel.digits or {}
6919     Babel.digits[\the\localeid] =
6920         table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6921     if not Babel.numbers then
6922         function Babel.numbers(head)
6923             local LOCALE = Babel.attr_locale
6924             local GLYPH = node.id'glyph'
6925             local inmath = false
6926             for item in node.traverse(head) do
6927                 if not inmath and item.id == GLYPH then
6928                     local temp = node.get_attribute(item, LOCALE)
6929                     if Babel.digits[temp] then
6930                         local chr = item.char
6931                         if chr > 47 and chr < 58 then
6932                             item.char = Babel.digits[temp][chr-47]
6933                         end
6934                     end
6935                     elseif item.id == node.id'math' then
6936                         inmath = (item.subtype == 0)
6937                     end
6938                 end
6939             return head
6940         end
6941     end
6942 }}%
6943 \fi
6944 % == transforms ==
6945 \ifx\bbl@KVP@transforms\@nnil\else
6946 \def\bbl@elt##1##2##3{%
6947     \in@{$transforms.}{##1}%
6948     \ifin@
6949         \def\bbl@tempa{##1}%
6950         \bbl@replace\bbl@tempa{transforms.}{}%
6951         \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6952     \fi}%
6953 \bbl@exp{%
6954     \\bbl@ifblank{\bbl@cl{dgnat}}%
6955     {\let\\bbl@tempa\relax}%
6956     {\def\\bbl@tempa{%
6957         \\bbl@elt{transforms.prehyphenation}%
6958         {digits.native.1.0}{([0-9])}%

```

```

6959      \\bbl@elt{transforms.prehyphenation}%
6960      {digits.native.1.1}{string={\string|0123456789\string|\bbl@cl{dgnat}}}}}%
6961 \ifx\bbl@tempa\relax\else
6962   \toks@\expandafter\expandafter\expandafter{%
6963     \csname bbl@inidata@\language\endcsname}%
6964     \bbl@csarg\edef{inidata@\language}%
6965     \unexpanded\expandafter{\bbl@tempa}%
6966     \the\toks@}%
6967 \fi
6968 \csname bbl@inidata@\language\endcsname
6969 \bbl@release@transforms\relax % \relax closes the last item.
6970 \fi}

```

Start tabular here:

```

6971 \def\localerestoredirs{%
6972   \ifcase\bbl@thetextdir
6973     \ifnum\textdirection=\z@ \else \textdir TLT \fi
6974   \else
6975     \ifnum\textdirection=\@ne \else \textdir TRT \fi
6976   \fi
6977   \ifcase\bbl@thepardir
6978     \ifnum\pardirection=\z@ \else \pardir TLT \bodydir TLT \fi
6979   \else
6980     \ifnum\pardirection=\@ne \else \pardir TRT \bodydir TRT \fi
6981   \fi}
6982 %
6983 \IfBabelLayout{tabular}%
6984   {\chardef\bbl@tabular@mode\tw@}% All RTL
6985   {\IfBabelLayout{notabular}%
6986     {\chardef\bbl@tabular@mode\z@}%
6987     {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6988 %
6989 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6990 % Redefine: vrules mess up dirs.
6991 \def\@arstrut{\relax\copy\@arstrutbox}%
6992 \ifcase\bbl@tabular@mode \or % 1 = Mixed - default
6993   \let\bbl@parabefore\relax
6994   \AddToHook{para/before}{\bbl@parabefore}
6995   \AtBeginDocument{%
6996     \bbl@replace\@tabular{${}$}%
6997     \def\bbl@insidemath{0}%
6998     \def\bbl@parabefore{\localerestoredirs}}%
6999   \ifnum\bbl@tabular@mode=\@ne
7000     \bbl@ifunset{\@tabclassz}{}%
7001     \bbl@exp{% Hide conditionals
7002       \\bbl@sreplace\\ \@tabclassz
7003       {\<ifcase>\\ \@chnum}%
7004       {\localerestoredirs\<ifcase>\\ \@chnum}}}%
7005     \@ifpackage{colortbl}%
7006     {\bbl@sreplace\@classz
7007       {\hbox\bgroup\bgroup{\hbox\bgroup\bgroup\localerestoredirs}}}%
7008     {\@ifpackage{array}%
7009       {\bbl@exp{% Hide conditionals
7010         \\bbl@sreplace\\ \@classz
7011         {\<ifcase>\\ \@chnum}%
7012         {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
7013         \\bbl@sreplace\\ \@classz
7014         {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
7015       {}}}%
7016   \fi}%
7017 \or % 2 = All RTL - tabular
7018   \let\bbl@parabefore\relax
7019   \AddToHook{para/before}{\bbl@parabefore}%

```



```

7020 \AtBeginDocument{%
7021 \ifpackageloaded{colortbl}%
7022 {\bbl@replace\@tabular{$}{$}%
7023 \def\bbl@insidemath{0}%
7024 \def\bbl@parabefore{\localerestoredirs}}%
7025 \bbl@sreplace\@classz
7026 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
7027 {}}%
7028 \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.

```

7029 \AtBeginDocument{%
7030 \ifpackageloaded{multicol}%
7031 {\toks@expandafter{\multi@column@out}%
7032 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
7033 {}%
7034 \ifpackageloaded{paracol}%
7035 {\edef\pcol@output{%
7036 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7037 {}}%
7038 \fi

```

Finish here if there in no layout.

```

7039 \ifx\bbl@opt@layout\@nnil\endinput\fi

```

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

```

7040 \ifnum\bbl@bidimode>\z@ % Any bidi=
7041 \def\bbl@nextfake#1% non-local changes, use always inside a group!
7042 \bbl@exp{%
7043 \mathdir\the\bodydir
7044 #1% Once entered in math, set boxes to restore values
7045 \def\bbbl@insidemath{0}%
7046 \<ifmmode>%
7047 \everyvbox{%
7048 \the\everyvbox
7049 \bodydir\the\bodydir
7050 \mathdir\the\mathdir
7051 \everyhbox{\the\everyhbox}%
7052 \everyvbox{\the\everyvbox}}%
7053 \everyhbox{%
7054 \the\everyhbox
7055 \bodydir\the\bodydir
7056 \mathdir\the\mathdir
7057 \everyhbox{\the\everyhbox}%
7058 \everyvbox{\the\everyvbox}}%
7059 \<fi>}}%
7060 \IfBabelLayout{nopars}
7061 {}
7062 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7063 \IfBabelLayout{pars}
7064 {\def\@hangfrom#1%
7065 \setbox\@tempboxa\hbox{#1}}%
7066 \hangindent\wd\@tempboxa
7067 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
7068 \shapemode\@ne
7069 \fi
7070 \noindent\box\@tempboxa}}
7071 {}

```

```

7072 \fi
7073 %
7074 \IfBabelLayout{tabular}
7075 {\let\bbL@OL@tabular\@tabular
7076 \bbL@replace\@tabular{$}{\bbL@nextfake$}%
7077 \let\bbL@NL@tabular\@tabular
7078 \AtBeginDocument{%
7079 \ifx\bbL@NL@tabular\@tabular\else
7080 \bbL@exp{\in{\bbL@nextfake}{\@tabular}}}%
7081 \ifin\else
7082 \bbL@replace\@tabular{$}{\bbL@nextfake$}%
7083 \fi
7084 \let\bbL@NL@tabular\@tabular
7085 \fi}}
7086 {}
7087 %
7088 \IfBabelLayout{lists}
7089 {\let\bbL@OL@list\list
7090 \bbL@sreplace\list{\parshape}{\bbL@listparshape}%
7091 \let\bbL@NL@list\list
7092 \def\bbL@listparshape#1#2#3{%
7093 \parshape #1 #2 #3 %
7094 \ifnum\bbL@getluadir{page}=\bbL@getluadir{par}\else
7095 \shapemode\tw@
7096 \fi}}
7097 {}
7098 %
7099 \IfBabelLayout{graphics}
7100 {\let\bbL@pictresetdir\relax
7101 \def\bbL@pictsetdir#1{%
7102 \ifcase\bbL@thetextdir
7103 \let\bbL@pictresetdir\relax
7104 \else
7105 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7106 \or\textdir TLT
7107 \else\bodydir TLT \textdir TLT
7108 \fi
7109 % \(\text|par)dir required in pgf:
7110 \def\bbL@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7111 \fi}%
7112 \AddToHook{env/picture/begin}{\bbL@pictsetdir\tw@}%
7113 \directlua{
7114 Babel.get_picture_dir = true
7115 Babel.picture_has_bidi = 0
7116 %
7117 function Babel.picture_dir (head)
7118 if not Babel.get_picture_dir then return head end
7119 if Babel.hlist_has_bidi(head) then
7120 Babel.picture_has_bidi = 1
7121 end
7122 return head
7123 end
7124 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7125 "Babel.picture_dir")
7126 }%
7127 \AtBeginDocument{%
7128 \def\LS@rot{%
7129 \setbox\@outputbox\vbox{%
7130 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
7131 \long\def\put(#1,#2)#3{%
7132 \@killglue
7133 % Try:
7134 \ifx\bbL@pictresetdir\relax

```

```

7135     \def\bbl@tempc{0}%
7136 \else
7137     \directlua{
7138         Babel.get_picture_dir = true
7139         Babel.picture_has_bidi = 0
7140     }%
7141     \setbox\z@\hb@xt@\z@{%
7142         \@defaultunitsset\@tempdimc{#1}\unitlength
7143         \kern\@tempdimc
7144         #3\hss}%
7145     \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7146 \fi
7147 % Do:
7148 \@defaultunitsset\@tempdimc{#2}\unitlength
7149 \raise\@tempdimc\hb@xt@\z@{%
7150     \@defaultunitsset\@tempdimc{#1}\unitlength
7151     \kern\@tempdimc
7152     {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
7153 \ignorespaces}%
7154 \MakeRobust\put}%
7155 \AtBeginDocument
7156 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
7157 \ifx\pgfpicture\undefined\else
7158     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
7159     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
7160     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
7161 \fi
7162 \ifx\tikzpicture\undefined\else
7163     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
7164     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
7165     \bbl@sreplace\tikz{\begingroup}\begingroup\bbl@pictsetdir\tw@}%
7166     \bbl@sreplace\tikzpicture{\begingroup}\begingroup\bbl@pictsetdir\tw@}%
7167 \fi
7168 \ifx\tcolorbox\undefined\else
7169     \def\tcb@drawing@env@begin{%
7170         \csname tcb@before@\tcb@split@state\endcsname
7171         \bbl@pictsetdir\tw@
7172         \begin{\kvtcb@graphenv}%
7173         \tcb@bbdraw
7174         \tcb@apply@graph@patches}%
7175     \def\tcb@drawing@env@end{%
7176         \end{\kvtcb@graphenv}%
7177         \bbl@pictresetdir
7178         \csname tcb@after@\tcb@split@state\endcsname}%
7179 \fi
7180 }}
7181 {}

```

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.

```

7182 \IfBabelLayout{counters*}%
7183 {\bbl@add\bbl@opt@layout{.counters.}%
7184 \directlua{
7185     luatexbase.add_to_callback("process_output_buffer",
7186         Babel.discard_sublr , "Babel.discard_sublr") }%
7187 {}%
7188 \IfBabelLayout{counters}%
7189 {\let\bbl@0L@textsuperscript\textsuperscript
7190 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7191 \let\bbl@latinarabic=\arabic
7192 \let\bbl@0L@arabic\arabic
7193 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%

```

```

7194 \@ifpackagewith{babel}{bidi=default}%
7195 {\let\bbl@asciroman=\@roman
7196 \let\bbl@OL@@roman\@roman
7197 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7198 \let\bbl@asciRoman=\@Roman
7199 \let\bbl@OL@@roman\@Roman
7200 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
7201 \let\bbl@OL@labelenumii\labelenumii
7202 \def\labelenumii{}\theenumii}%
7203 \let\bbl@OL@p@enumiii\p@enumiii
7204 \def\p@enumiii{\p@enumii}\theenumii{}\{}\{}\}

```

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

```

7205 \IfBabelLayout{extras}%
7206 {\bbl@ncarg\let\bbl@OL@underline{underline }%
7207 \bbl@carg\bbl@sreplace{underline }%
7208 {\$@@@underline}\bgroup\bbl@nextfake$@@@underline}%
7209 \bbl@carg\bbl@sreplace{underline }%
7210 {\m@th$}\m@th$\egroup}%
7211 \let\bbl@OL@LaTeXe\LaTeXe
7212 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7213 \if b\expandafter\@car\f@series\@nil\boldmath\fi
7214 \babelsublr{%
7215 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
7216 {}
7217 </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.

```

7218 <{*transforms}>
7219 Babel.linebreaking.replacements = {}
7220 Babel.linebreaking.replacements[0] = {} -- pre
7221 Babel.linebreaking.replacements[1] = {} -- post
7222
7223 function Babel.tovalue(v)
7224 if type(v) == 'table' then
7225     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7226 else
7227     return v
7228 end
7229 end
7230
7231 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr_hboxed'
7232
7233 function Babel.set_hboxed(head, gc)
7234 for item in node.traverse(head) do
7235     node.set_attribute(item, Babel.attr_hboxed, 1)
7236 end
7237 return head
7238 end
7239

```

```

7240 Babel.fetch_subtext = {}
7241
7242 Babel.ignore_pre_char = function(node)
7243   return (node.lang == Babel.nohyphenation)
7244 end
7245
7246 Babel.show_transforms = false
7247
7248 -- Merging both functions doesn't seem feasible, because there are too
7249 -- many differences.
7250 Babel.fetch_subtext[0] = function(head)
7251   local word_string = ''
7252   local word_nodes = {}
7253   local lang
7254   local item = head
7255   local inmath = false
7256
7257   while item do
7258
7259     if item.id == 11 then
7260       inmath = (item.subtype == 0)
7261     end
7262
7263     if inmath then
7264       -- pass
7265
7266     elseif item.id == 29 then
7267       local locale = node.get_attribute(item, Babel.attr_locale)
7268
7269       if lang == locale or lang == nil then
7270         lang = lang or locale
7271         if Babel.ignore_pre_char(item) then
7272           word_string = word_string .. Babel.us_char
7273         else
7274           if node.has_attribute(item, Babel.attr_hboxed) then
7275             word_string = word_string .. Babel.us_char
7276           else
7277             word_string = word_string .. unicode.utf8.char(item.char)
7278           end
7279         end
7280         word_nodes[#word_nodes+1] = item
7281       else
7282         break
7283       end
7284
7285     elseif item.id == 12 and item.subtype == 13 then
7286       if node.has_attribute(item, Babel.attr_hboxed) then
7287         word_string = word_string .. Babel.us_char
7288       else
7289         word_string = word_string .. ' '
7290       end
7291       word_nodes[#word_nodes+1] = item
7292
7293       -- Ignore leading unrecognized nodes, too.
7294     elseif word_string ~= '' then
7295       word_string = word_string .. Babel.us_char
7296       word_nodes[#word_nodes+1] = item -- Will be ignored
7297     end
7298
7299     item = item.next
7300   end
7301
7302   -- Here and above we remove some trailing chars but not the

```

```

7303 -- corresponding nodes. But they aren't accessed.
7304 if word_string:sub(-1) == ' ' then
7305     word_string = word_string:sub(1,-2)
7306 end
7307 if Babel.show_transforms then texio.write_nl(word_string) end
7308 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7309 return word_string, word_nodes, item, lang
7310 end
7311
7312 Babel.fetch_subtext[1] = function(head)
7313     local word_string = ''
7314     local word_nodes = {}
7315     local lang
7316     local item = head
7317     local inmath = false
7318
7319     while item do
7320
7321         if item.id == 11 then
7322             inmath = (item.subtype == 0)
7323         end
7324
7325         if inmath then
7326             -- pass
7327
7328         elseif item.id == 29 then
7329             if item.lang == lang or lang == nil then
7330                 lang = lang or item.lang
7331                 if node.has_attribute(item, Babel.attr_hboxed) then
7332                     word_string = word_string .. Babel.us_char
7333                 elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7334                     word_string = word_string .. Babel.us_char
7335                 else
7336                     word_string = word_string .. unicode.utf8.char(item.char)
7337                 end
7338                 word_nodes[#word_nodes+1] = item
7339             else
7340                 break
7341             end
7342
7343         elseif item.id == 7 and item.subtype == 2 then
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 .. '='
7348             end
7349             word_nodes[#word_nodes+1] = item
7350
7351         elseif item.id == 7 and item.subtype == 3 then
7352             if node.has_attribute(item, Babel.attr_hboxed) then
7353                 word_string = word_string .. Babel.us_char
7354             else
7355                 word_string = word_string .. '|'
7356             end
7357             word_nodes[#word_nodes+1] = item
7358
7359         -- (1) Go to next word if nothing was found, and (2) implicitly
7360         -- remove leading USs.
7361         elseif word_string == '' then
7362             -- pass
7363
7364         -- This is the responsible for splitting by words.
7365         elseif (item.id == 12 and item.subtype == 13) then

```

```

7366     break
7367
7368     else
7369         word_string = word_string .. Babel.us_char
7370         word_nodes[#word_nodes+1] = item -- Will be ignored
7371     end
7372
7373     item = item.next
7374 end
7375 if Babel.show_transforms then texio.write_nl(word_string) end
7376 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7377 return word_string, word_nodes, item, lang
7378 end
7379
7380 function Babel.pre_hyphenate_replace(head)
7381     Babel.hyphenate_replace(head, 0)
7382 end
7383
7384 function Babel.post_hyphenate_replace(head)
7385     Babel.hyphenate_replace(head, 1)
7386 end
7387
7388 Babel.us_char = string.char(31)
7389
7390 function Babel.hyphenate_replace(head, mode)
7391     local u = unicode.utf8
7392     local lbkr = Babel.linebreaking.replacements[mode]
7393     local tovalue = Babel.tovalue
7394
7395     local word_head = head
7396
7397     if Babel.show_transforms then
7398         texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7399     end
7400
7401     while true do -- for each subtext block
7402
7403         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7404
7405         if Babel.debug then
7406             print()
7407             print((mode == 0) and '@@@@<' or '@@@@>', w)
7408         end
7409
7410         if nw == nil and w == '' then break end
7411
7412         if not lang then goto next end
7413         if not lbkr[lang] then goto next end
7414
7415         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7416         -- loops are nested.
7417         for k=1, #lbkr[lang] do
7418             local p = lbkr[lang][k].pattern
7419             local r = lbkr[lang][k].replace
7420             local attr = lbkr[lang][k].attr or -1
7421
7422             if Babel.debug then
7423                 print('*****', p, mode)
7424             end
7425
7426             -- This variable is set in some cases below to the first *byte*
7427             -- after the match, either as found by u.match (faster) or the
7428             -- computed position based on sc if w has changed.

```

```

7429     local last_match = 0
7430     local step = 0
7431
7432     -- For every match.
7433     while true do
7434         if Babel.debug then
7435             print('====')
7436         end
7437         local new -- used when inserting and removing nodes
7438         local dummy_node -- used by after
7439
7440         local matches = { u.match(w, p, last_match) }
7441
7442         if #matches < 2 then break end
7443
7444         -- Get and remove empty captures (with ()'s, which return a
7445         -- number with the position), and keep actual captures
7446         -- (from (...)), if any, in matches.
7447         local first = table.remove(matches, 1)
7448         local last = table.remove(matches, #matches)
7449         -- Non re-fetched substrings may contain \31, which separates
7450         -- subsubstrings.
7451         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7452
7453         local save_last = last -- with A()BC()D, points to D
7454
7455         -- Fix offsets, from bytes to unicode. Explained above.
7456         first = u.len(w:sub(1, first-1)) + 1
7457         last = u.len(w:sub(1, last-1)) -- now last points to C
7458
7459         -- This loop stores in a small table the nodes
7460         -- corresponding to the pattern. Used by 'data' to provide a
7461         -- predictable behavior with 'insert' (w_nodes is modified on
7462         -- the fly), and also access to 'remove'd nodes.
7463         local sc = first-1 -- Used below, too
7464         local data_nodes = {}
7465
7466         local enabled = true
7467         for q = 1, last-first+1 do
7468             data_nodes[q] = w_nodes[sc+q]
7469             if enabled
7470                 and attr > -1
7471                 and not node.has_attribute(data_nodes[q], attr)
7472             then
7473                 enabled = false
7474             end
7475         end
7476
7477         -- This loop traverses the matched substring and takes the
7478         -- corresponding action stored in the replacement list.
7479         -- sc = the position in substr nodes / string
7480         -- rc = the replacement table index
7481         local rc = 0
7482
7483         ----- TODO. dummy_node?
7484         while rc < last-first+1 or dummy_node do -- for each replacement
7485             if Babel.debug then
7486                 print('.....', rc + 1)
7487             end
7488             sc = sc + 1
7489             rc = rc + 1
7490
7491             if Babel.debug then

```



```

7492     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7493     local ss = ''
7494     for itt in node.traverse(head) do
7495         if itt.id == 29 then
7496             ss = ss .. unicode.utf8.char(itt.char)
7497         else
7498             ss = ss .. '{' .. itt.id .. '}'
7499         end
7500     end
7501     print('*****', ss)
7502
7503 end
7504
7505 local crep = r[rc]
7506 local item = w_nodes[sc]
7507 local item_base = item
7508 local placeholder = Babel.us_char
7509 local d
7510
7511 if crep and crep.data then
7512     item_base = data_nodes[crep.data]
7513 end
7514
7515 if crep then
7516     step = crep.step or step
7517 end
7518
7519 if crep and crep.after then
7520     crep.insert = true
7521     if dummy_node then
7522         item = dummy_node
7523     else -- TODO. if there is a node after?
7524         d = node.copy(item_base)
7525         head, item = node.insert_after(head, item, d)
7526         dummy_node = item
7527     end
7528 end
7529
7530 if crep and not crep.after and dummy_node then
7531     node.remove(head, dummy_node)
7532     dummy_node = nil
7533 end
7534
7535 if not enabled then
7536     last_match = save_last
7537     goto next
7538
7539 elseif crep and next(crep) == nil then -- = {}
7540     if step == 0 then
7541         last_match = save_last -- Optimization
7542     else
7543         last_match = utf8.offset(w, sc+step)
7544     end
7545     goto next
7546
7547 elseif crep == nil or crep.remove then
7548     node.remove(head, item)
7549     table.remove(w_nodes, sc)
7550     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7551     sc = sc - 1 -- Nothing has been inserted.
7552     last_match = utf8.offset(w, sc+1+step)
7553     goto next
7554

```

```

7555     elseif crep and crep.kashida then -- Experimental
7556         node.set_attribute(item,
7557             Babel.attr_kashida,
7558             crep.kashida)
7559         last_match = utf8.offset(w, sc+1+step)
7560         goto next
7561
7562     elseif crep and crep.string then
7563         local str = crep.string(matches)
7564         if str == '' then -- Gather with nil
7565             node.remove(head, item)
7566             table.remove(w_nodes, sc)
7567             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7568             sc = sc - 1 -- Nothing has been inserted.
7569         else
7570             local loop_first = true
7571             for s in string.utfvalues(str) do
7572                 d = node.copy(item_base)
7573                 d.char = s
7574                 if loop_first then
7575                     loop_first = false
7576                     head, new = node.insert_before(head, item, d)
7577                     if sc == 1 then
7578                         word_head = head
7579                     end
7580                     w_nodes[sc] = d
7581                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7582                 else
7583                     sc = sc + 1
7584                     head, new = node.insert_before(head, item, d)
7585                     table.insert(w_nodes, sc, new)
7586                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7587                 end
7588                 if Babel.debug then
7589                     print('.....', 'str')
7590                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7591                 end
7592             end -- for
7593             node.remove(head, item)
7594         end -- if ''
7595         last_match = utf8.offset(w, sc+1+step)
7596         goto next
7597
7598     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7599         d = node.new(7, 3) -- (disc, regular)
7600         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7601         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7602         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7603         d.attr = item_base.attr
7604         if crep.pre == nil then -- TeXbook p96
7605             d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7606         else
7607             d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7608         end
7609         placeholder = '|'
7610         head, new = node.insert_before(head, item, d)
7611
7612     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7613         -- ERROR
7614
7615     elseif crep and crep.penalty then
7616         d = node.new(14, 0) -- (penalty, userpenalty)
7617         d.attr = item_base.attr

```

```

7618         d.penalty = tovalue(crep.penalty)
7619         head, new = node.insert_before(head, item, d)
7620
7621     elseif crep and crep.space then
7622         -- 655360 = 10 pt = 10 * 65536 sp
7623         d = node.new(12, 13)      -- (glue, spaceskip)
7624         local quad = font.getfont(item_base.font).size or 655360
7625         node.setglue(d, tovalue(crep.space[1]) * quad,
7626                       tovalue(crep.space[2]) * quad,
7627                       tovalue(crep.space[3]) * quad)
7628         if mode == 0 then
7629             placeholder = ' '
7630         end
7631         head, new = node.insert_before(head, item, d)
7632
7633     elseif crep and crep.norule then
7634         -- 655360 = 10 pt = 10 * 65536 sp
7635         d = node.new(2, 3)      -- (rule, empty) = \no*rule
7636         local quad = font.getfont(item_base.font).size or 655360
7637         d.width  = tovalue(crep.norule[1]) * quad
7638         d.height = tovalue(crep.norule[2]) * quad
7639         d.depth  = tovalue(crep.norule[3]) * quad
7640         head, new = node.insert_before(head, item, d)
7641
7642     elseif crep and crep.spacefactor then
7643         d = node.new(12, 13)      -- (glue, spaceskip)
7644         local base_font = font.getfont(item_base.font)
7645         node.setglue(d,
7646                     tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7647                     tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7648                     tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7649         if mode == 0 then
7650             placeholder = ' '
7651         end
7652         head, new = node.insert_before(head, item, d)
7653
7654     elseif mode == 0 and crep and crep.space then
7655         -- ERROR
7656
7657     elseif crep and crep.kern then
7658         d = node.new(13, 1)      -- (kern, user)
7659         local quad = font.getfont(item_base.font).size or 655360
7660         d.attr = item_base.attr
7661         d.kern = tovalue(crep.kern) * quad
7662         head, new = node.insert_before(head, item, d)
7663
7664     elseif crep and crep.node then
7665         d = node.new(crep.node[1], crep.node[2])
7666         d.attr = item_base.attr
7667         head, new = node.insert_before(head, item, d)
7668
7669     end -- i.e., replacement cases
7670
7671     -- Shared by disc, space(factor), kern, node and penalty.
7672     if sc == 1 then
7673         word_head = head
7674     end
7675     if crep.insert then
7676         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7677         table.insert(w_nodes, sc, new)
7678         last = last + 1
7679     else
7680         w_nodes[sc] = d

```

```

7681         node.remove(head, item)
7682         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7683     end
7684
7685     last_match = utf8.offset(w, sc+1+step)
7686
7687     ::next::
7688
7689     end -- for each replacement
7690
7691     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7692     if Babel.debug then
7693         print('.....', '/')
7694         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7695     end
7696
7697     if dummy_node then
7698         node.remove(head, dummy_node)
7699         dummy_node = nil
7700     end
7701
7702     end -- for match
7703
7704     end -- for patterns
7705
7706     ::next::
7707     word_head = nw
7708     end -- for substring
7709
7710     if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7711     return head
7712 end
7713
7714 -- This table stores capture maps, numbered consecutively
7715 Babel.capture_maps = {}
7716
7717 function Babel.esc_hex_to_char(h)
7718     if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7719         tex.getcatcode(tonumber(h, 16)) ~= 12 then
7720         return string.format([[Uchar"%X ]], tonumber(h,16))
7721     else
7722         return unicode.utf8.char(tonumber(h, 16))
7723     end
7724 end
7725
7726 -- The following functions belong to the next macro
7727 function Babel.capture_func(key, cap)
7728     local ret = "[[" .. cap:gsub('{{([0-9])}}', ")]..m[%1]..[["] .. "]"
7729     local cnt
7730     local u = unicode.utf8
7731     ret = u.gsub(ret, '{(%x%x%x%x+)}', '\x01%\x04')
7732     ret, cnt = ret:gsub('{{([0-9])|([^\^]|+)|(.-)}}', Babel.capture_func_map)
7733     ret = u.gsub(ret, '\x01(%x%x%x%x+)\x04', Babel.esc_hex_to_char)
7734     ret = ret:gsub("%[%]%%.%.%", '')
7735     ret = ret:gsub("%.%.%[%]%%", '')
7736     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7737 end
7738
7739 function Babel.capt_map(from, mapno)
7740     return Babel.capture_maps[mapno][from] or from
7741 end
7742
7743 -- Handle the {n|abc|ABC} syntax in captures

```

```

7744 function Babel.capture_func_map(capno, from, to)
7745     local u = unicode.utf8
7746     from = u.gsub(from, '\x01(%x%x%x%x+)\x04',
7747         function (n)
7748             return u.char(tonumber(n, 16))
7749         end)
7750     to = u.gsub(to, '\x01(%x%x%x%x+)\x04',
7751         function (n)
7752             return u.char(tonumber(n, 16))
7753         end)
7754     local froms = {}
7755     for s in string.utfcharacters(from) do
7756         table.insert(froms, s)
7757     end
7758     local cnt = 1
7759     table.insert(Babel.capture_maps, {})
7760     local mlen = table.getn(Babel.capture_maps)
7761     for s in string.utfcharacters(to) do
7762         Babel.capture_maps[mlen][froms[cnt]] = s
7763         cnt = cnt + 1
7764     end
7765     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7766         (mlen) .. " ).." .. "[["
7767 end
7768
7769 -- Create/Extend reversed sorted list of kashida weights:
7770 function Babel.capture_kashida(key, wt)
7771     wt = tonumber(wt)
7772     if Babel.kashida_wts then
7773         for p, q in ipairs(Babel.kashida_wts) do
7774             if wt == q then
7775                 break
7776             elseif wt > q then
7777                 table.insert(Babel.kashida_wts, p, wt)
7778                 break
7779             elseif table.getn(Babel.kashida_wts) == p then
7780                 table.insert(Babel.kashida_wts, wt)
7781             end
7782         end
7783     else
7784         Babel.kashida_wts = { wt }
7785     end
7786     return 'kashida = ' .. wt
7787 end
7788
7789 function Babel.capture_node(id, subtype)
7790     local sbt = 0
7791     for k, v in pairs(node.subtypes(id)) do
7792         if v == subtype then sbt = k end
7793     end
7794     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7795 end
7796
7797 -- Experimental: applies prehyphenation transforms to a string (letters
7798 -- and spaces).
7799 function Babel.string_prehyphenation(str, locale)
7800     local n, head, last, res
7801     head = node.new(8, 0) -- dummy (hack just to start)
7802     last = head
7803     for s in string.utfvalues(str) do
7804         if s == 20 then
7805             n = node.new(12, 0)
7806         else

```

```

7807     n = node.new(29, 0)
7808     n.char = s
7809 end
7810 node.set_attribute(n, Babel.attr_locale, locale)
7811 last.next = n
7812 last = n
7813 end
7814 head = Babel.hyphenate_replace(head, 0)
7815 res = ''
7816 for n in node.traverse(head) do
7817   if n.id == 12 then
7818     res = res .. ' '
7819   elseif n.id == 29 then
7820     res = res .. unicode.utf8.char(n.char)
7821   end
7822 end
7823 tex.print(res)
7824 end
7825 </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.

```

7826 <{*basic-r>
7827 Babel.bidi_enabled = true
7828
7829 require('babel-data-bidi.lua')

```

```

7830
7831 local characters = Babel.characters
7832 local ranges = Babel.ranges
7833
7834 local DIR = node.id("dir")
7835
7836 local function dir_mark(head, from, to, outer)
7837   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7838   local d = node.new(DIR)
7839   d.dir = '+' .. dir
7840   node.insert_before(head, from, d)
7841   d = node.new(DIR)
7842   d.dir = '-' .. dir
7843   node.insert_after(head, to, d)
7844 end
7845
7846 function Babel.bidi(head, ispar)
7847   local first_n, last_n          -- first and last char with nums
7848   local last_es                 -- an auxiliary 'last' used with nums
7849   local first_d, last_d         -- first and last char in L/R block
7850   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):

```

7851   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7852   local strong_lr = (strong == 'l') and 'l' or 'r'
7853   local outer = strong
7854
7855   local new_dir = false
7856   local first_dir = false
7857   local inmath = false
7858
7859   local last_lr
7860
7861   local type_n = ''
7862
7863   for item in node.traverse(head) do
7864
7865     -- three cases: glyph, dir, otherwise
7866     if item.id == node.id'glyph'
7867       or (item.id == 7 and item.subtype == 2) then
7868
7869       local itemchar
7870       if item.id == 7 and item.subtype == 2 then
7871         itemchar = item.replace.char
7872       else
7873         itemchar = item.char
7874       end
7875       local chardata = characters[itemchar]
7876       dir = chardata and chardata.d or nil
7877       if not dir then
7878         for nn, et in ipairs(ranges) do
7879           if itemchar < et[1] then
7880             break
7881           elseif itemchar <= et[2] then
7882             dir = et[3]
7883             break
7884           end
7885         end
7886       end
7887       dir = dir or 'l'
7888       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).

```

7889     if new_dir then
7890         attr_dir = 0
7891         for at in node.traverse(item.attr) do
7892             if at.number == Babel.attr_dir then
7893                 attr_dir = at.value & 0x3
7894             end
7895         end
7896         if attr_dir == 1 then
7897             strong = 'r'
7898         elseif attr_dir == 2 then
7899             strong = 'al'
7900         else
7901             strong = 'l'
7902         end
7903         strong_lr = (strong == 'l') and 'l' or 'r'
7904         outer = strong_lr
7905         new_dir = false
7906     end
7907
7908     if dir == 'nsm' then dir = strong end          -- W1

```

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

```

7909     dir_real = dir          -- We need dir_real to set strong below
7910     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:

```

7911     if strong == 'al' then
7912         if dir == 'en' then dir = 'an' end          -- W2
7913         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7914         strong_lr = 'r'                             -- W3
7915     end

```

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

```

7916     elseif item.id == node.id'dir' and not inmath then
7917         new_dir = true
7918         dir = nil
7919     elseif item.id == node.id'math' then
7920         inmath = (item.subtype == 0)
7921     else
7922         dir = nil          -- Not a char
7923     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>.

```

7924     if dir == 'en' or dir == 'an' or dir == 'et' then
7925         if dir ~= 'et' then
7926             type_n = dir
7927         end
7928         first_n = first_n or item
7929         last_n = last_es or item
7930         last_es = nil
7931     elseif dir == 'es' and last_n then -- W3+W6
7932         last_es = item
7933     elseif dir == 'cs' then          -- it's right - do nothing

```



```

7934     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7935         if strong_lr == 'r' and type_n ~= '' then
7936             dir_mark(head, first_n, last_n, 'r')
7937         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7938             dir_mark(head, first_n, last_n, 'r')
7939             dir_mark(head, first_d, last_d, outer)
7940             first_d, last_d = nil, nil
7941         elseif strong_lr == 'l' and type_n ~= '' then
7942             last_d = last_n
7943         end
7944         type_n = ''
7945         first_n, last_n = nil, nil
7946     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:

```

7947     if dir == 'l' or dir == 'r' then
7948         if dir ~= outer then
7949             first_d = first_d or item
7950             last_d = item
7951         elseif first_d and dir ~= strong_lr then
7952             dir_mark(head, first_d, last_d, outer)
7953             first_d, last_d = nil, nil
7954         end
7955     end

```

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

```

7956     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7957         item.char = characters[item.char] and
7958             characters[item.char].m or item.char
7959     elseif (dir or new_dir) and last_lr ~= item then
7960         local mir = outer .. strong_lr .. (dir or outer)
7961         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7962             for ch in node.traverse(node.next(last_lr)) do
7963                 if ch == item then break end
7964                 if ch.id == node.id'glyph' and characters[ch.char] then
7965                     ch.char = characters[ch.char].m or ch.char
7966                 end
7967             end
7968         end
7969     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).

```

7970     if dir == 'l' or dir == 'r' then
7971         last_lr = item
7972         strong = dir_real -- Don't search back - best save now
7973         strong_lr = (strong == 'l') and 'l' or 'r'
7974     elseif new_dir then
7975         last_lr = nil
7976     end
7977 end

```

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

```

7978     if last_lr and outer == 'r' then
7979         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7980             if characters[ch.char] then
7981                 ch.char = characters[ch.char].m or ch.char

```

```

7982     end
7983   end
7984 end
7985 if first_n then
7986   dir_mark(head, first_n, last_n, outer)
7987 end
7988 if first_d then
7989   dir_mark(head, first_d, last_d, outer)
7990 end

```

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

```

7991 return node.prev(head) or head
7992 end
7993 ⟨/basic-r⟩

```

And here the Lua code for bidi=basic:

```

7994 ⟨*basic⟩
7995 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7996
7997 Babel.fontmap = Babel.fontmap or {}
7998 Babel.fontmap[0] = {}      -- l
7999 Babel.fontmap[1] = {}      -- r
8000 Babel.fontmap[2] = {}      -- al/an
8001
8002 -- To cancel mirroring. Also OML, OMS, U?
8003 Babel.symbol_fonts = Babel.symbol_fonts or {}
8004 Babel.symbol_fonts[font.id('tenln')] = true
8005 Babel.symbol_fonts[font.id('tenlnw')] = true
8006 Babel.symbol_fonts[font.id('tencirc')] = true
8007 Babel.symbol_fonts[font.id('tencircw')] = true
8008
8009 Babel.bidi_enabled = true
8010 Babel.mirroring_enabled = true
8011
8012 require('babel-data-bidi.lua')
8013
8014 local characters = Babel.characters
8015 local ranges = Babel.ranges
8016
8017 local DIR = node.id('dir')
8018 local GLYPH = node.id('glyph')
8019
8020 local function insert_implicit(head, state, outer)
8021   local new_state = state
8022   if state.sim and state.eim and state.sim ~= state.eim then
8023     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
8024     local d = node.new(DIR)
8025     d.dir = '+' .. dir
8026     node.insert_before(head, state.sim, d)
8027     local d = node.new(DIR)
8028     d.dir = '-' .. dir
8029     node.insert_after(head, state.eim, d)
8030   end
8031   new_state.sim, new_state.eim = nil, nil
8032   return head, new_state
8033 end
8034
8035 local function insert_numeric(head, state)
8036   local new
8037   local new_state = state
8038   if state.san and state.ean and state.san ~= state.ean then
8039     local d = node.new(DIR)
8040     d.dir = '+TLT'

```

```

8041     _, new = node.insert_before(head, state.san, d)
8042     if state.san == state.sim then state.sim = new end
8043     local d = node.new(DIR)
8044     d.dir = '-TLT'
8045     _, new = node.insert_after(head, state.ean, d)
8046     if state.ean == state.eim then state.eim = new end
8047 end
8048 new_state.san, new_state.ean = nil, nil
8049 return head, new_state
8050 end
8051
8052 local function glyph_not_symbol_font(node)
8053   if node.id == GLYPH then
8054     return not Babel.symbol_fonts[node.font]
8055   else
8056     return false
8057   end
8058 end
8059
8060 -- TODO - \hbox with an explicit dir can lead to wrong results
8061 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
8062 -- was made to improve the situation, but the problem is the 3-dir
8063 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8064 -- well.
8065
8066 function Babel.bidi(head, ispar, hdir)
8067   local d -- d is used mainly for computations in a loop
8068   local prev_d = ''
8069   local new_d = false
8070
8071   local nodes = {}
8072   local outer_first = nil
8073   local inmath = false
8074
8075   local glue_d = nil
8076   local glue_i = nil
8077
8078   local has_en = false
8079   local first_et = nil
8080
8081   local has_hyperlink = false
8082
8083   local ATDIR = Babel.attr_dir
8084   local attr_d, temp
8085   local locale_d
8086
8087   local save_outer
8088   local locale_d = node.get_attribute(head, ATDIR)
8089   if locale_d then
8090     locale_d = locale_d & 0x3
8091     save_outer = (locale_d == 0 and 'l') or
8092                 (locale_d == 1 and 'r') or
8093                 (locale_d == 2 and 'al')
8094   elseif ispar then -- Or error? Shouldn't happen
8095     -- when the callback is called, we are just _after_ the box,
8096     -- and the textdir is that of the surrounding text
8097     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8098   else -- Empty box
8099     save_outer = ('TRT' == hdir) and 'r' or 'l'
8100   end
8101   local outer = save_outer
8102   local last = outer
8103   -- 'al' is only taken into account in the first, current loop

```

```

8104 if save_outer == 'al' then save_outer = 'r' end
8105
8106 local fontmap = Babel.fontmap
8107
8108 for item in node.traverse(head) do
8109     -- Mask: DxxxPPTT (Done, Paddir [0-2], Textdir [0-2])
8110     locale_d = node.get_attribute(item, ATDIR)
8111     node.set_attribute(item, ATDIR, 0x80)
8112
8113     -- In what follows, #node is the last (previous) node, because the
8114     -- current one is not added until we start processing the neutrals.
8115     -- three cases: glyph, dir, otherwise
8116     if glyph_not_symbol_font(item)
8117         or (item.id == 7 and item.subtype == 2) then
8118
8119         if locale_d == 0x80 then goto nextnode end
8120
8121         local d_font = nil
8122         local item_r
8123         if item.id == 7 and item.subtype == 2 then
8124             item_r = item.replace -- automatic discs have just 1 glyph
8125         else
8126             item_r = item
8127         end
8128
8129         local chardata = characters[item_r.char]
8130         d = chardata and chardata.d or nil
8131         if not d or d == 'nsm' then
8132             for nn, et in ipairs(ranges) do
8133                 if item_r.char < et[1] then
8134                     break
8135                 elseif item_r.char <= et[2] then
8136                     if not d then d = et[3]
8137                     elseif d == 'nsm' then d_font = et[3]
8138                     end
8139                     break
8140                 end
8141             end
8142         end
8143         d = d or 'l'
8144
8145         -- A short 'pause' in bidi for mapfont
8146         -- %%% TODO. move if fontmap here
8147         d_font = d_font or d
8148         d_font = (d_font == 'l' and 0) or
8149             (d_font == 'nsm' and 0) or
8150             (d_font == 'r' and 1) or
8151             (d_font == 'al' and 2) or
8152             (d_font == 'an' and 2) or nil
8153         if d_font and fontmap and fontmap[d_font][item_r.font] then
8154             item_r.font = fontmap[d_font][item_r.font]
8155         end
8156
8157         if new_d then
8158             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8159             if inmath then
8160                 attr_d = 0
8161             else
8162                 attr_d = locale_d & 0x3
8163             end
8164             if attr_d == 1 then
8165                 outer_first = 'r'
8166             end
8167         end
8168     end
8169 end

```

```

8167         last = 'r'
8168     elseif attr_d == 2 then
8169         outer_first = 'r'
8170         last = 'al'
8171     else
8172         outer_first = 'l'
8173         last = 'l'
8174     end
8175     outer = last
8176     has_en = false
8177     first_et = nil
8178     new_d = false
8179 end
8180
8181 if glue_d then
8182     if (d == 'l' and 'l' or 'r') ~= glue_d then
8183         table.insert(nodes, {glue_i, 'on', nil})
8184     end
8185     glue_d = nil
8186     glue_i = nil
8187 end
8188
8189 elseif item.id == DIR then
8190     d = nil
8191     new_d = true
8192
8193 elseif item.id == node.id'glue' and item.subtype == 13 then
8194     glue_d = d
8195     glue_i = item
8196     d = nil
8197
8198 elseif item.id == node.id'math' then
8199     inmath = (item.subtype == 0)
8200
8201 elseif item.id == 8 and item.subtype == 19 then
8202     has_hyperlink = true
8203
8204 else
8205     d = nil
8206 end
8207
8208 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8209 if last == 'al' and d == 'en' then
8210     d = 'an'           -- W3
8211 elseif last == 'al' and (d == 'et' or d == 'es') then
8212     d = 'on'           -- W6
8213 end
8214
8215 -- EN + CS/ES + EN      -- W4
8216 if d == 'en' and #nodes >= 2 then
8217     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8218         and nodes[#nodes-1][2] == 'en' then
8219         nodes[#nodes][2] = 'en'
8220     end
8221 end
8222
8223 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
8224 if d == 'an' and #nodes >= 2 then
8225     if (nodes[#nodes][2] == 'cs')
8226         and nodes[#nodes-1][2] == 'an' then
8227         nodes[#nodes][2] = 'an'
8228     end
8229 end

```

```

8230
8231 -- ET/EN -- W5 + W7->l / W6->on
8232 if d == 'et' then
8233     first_et = first_et or (#nodes + 1)
8234 elseif d == 'en' then
8235     has_en = true
8236     first_et = first_et or (#nodes + 1)
8237 elseif first_et then -- d may be nil here !
8238     if has_en then
8239         if last == 'l' then
8240             temp = 'l' -- W7
8241         else
8242             temp = 'en' -- W5
8243         end
8244     else
8245         temp = 'on' -- W6
8246     end
8247     for e = first_et, #nodes do
8248         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8249     end
8250     first_et = nil
8251     has_en = false
8252 end
8253
8254 -- Force mathdir in math if ON (currently works as expected only
8255 -- with 'l')
8256
8257 if inmath and d == 'on' then
8258     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8259 end
8260
8261 if d then
8262     if d == 'al' then
8263         d = 'r'
8264         last = 'al'
8265     elseif d == 'l' or d == 'r' then
8266         last = d
8267     end
8268     prev_d = d
8269     table.insert(nodes, {item, d, outer_first})
8270 end
8271
8272 outer_first = nil
8273
8274 ::nextnode::
8275
8276 end -- for each node
8277
8278 -- TODO -- repeated here in case EN/ET is the last node. Find a
8279 -- better way of doing things:
8280 if first_et then -- dir may be nil here !
8281     if has_en then
8282         if last == 'l' then
8283             temp = 'l' -- W7
8284         else
8285             temp = 'en' -- W5
8286         end
8287     else
8288         temp = 'on' -- W6
8289     end
8290     for e = first_et, #nodes do
8291         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8292     end

```

```

8293 end
8294
8295 -- dummy node, to close things
8296 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8297
8298 ----- NEUTRAL -----
8299
8300 outer = save_outer
8301 last = outer
8302
8303 local first_on = nil
8304
8305 for q = 1, #nodes do
8306     local item
8307
8308     local outer_first = nodes[q][3]
8309     outer = outer_first or outer
8310     last = outer_first or last
8311
8312     local d = nodes[q][2]
8313     if d == 'an' or d == 'en' then d = 'r' end
8314     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8315
8316     if d == 'on' then
8317         first_on = first_on or q
8318     elseif first_on then
8319         if last == d then
8320             temp = d
8321         else
8322             temp = outer
8323         end
8324         for r = first_on, q - 1 do
8325             nodes[r][2] = temp
8326             item = nodes[r][1] -- MIRRORING
8327             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8328                 and temp == 'r' and characters[item.char] then
8329                 local font_mode = ''
8330                 if item.font > 0 and font.fonts[item.font].properties then
8331                     font_mode = font.fonts[item.font].properties.mode
8332                 end
8333                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8334                     item.char = characters[item.char].m or item.char
8335                 end
8336             end
8337         end
8338         first_on = nil
8339     end
8340
8341     if d == 'r' or d == 'l' then last = d end
8342 end
8343
8344 ----- IMPLICIT, REORDER -----
8345
8346 outer = save_outer
8347 last = outer
8348
8349 local state = {}
8350 state.has_r = false
8351
8352 for q = 1, #nodes do
8353     local item = nodes[q][1]
8354
8355

```

```

8356     outer = nodes[q][3] or outer
8357
8358     local d = nodes[q][2]
8359
8360     if d == 'nsm' then d = last end          -- W1
8361     if d == 'en' then d = 'an' end
8362     local isdir = (d == 'r' or d == 'l')
8363
8364     if outer == 'l' and d == 'an' then
8365         state.san = state.san or item
8366         state.ean = item
8367     elseif state.san then
8368         head, state = insert_numeric(head, state)
8369     end
8370
8371     if outer == 'l' then
8372         if d == 'an' or d == 'r' then      -- im -> implicit
8373             if d == 'r' then state.has_r = true end
8374             state.sim = state.sim or item
8375             state.eim = item
8376         elseif d == 'l' and state.sim and state.has_r then
8377             head, state = insert_implicit(head, state, outer)
8378         elseif d == 'l' then
8379             state.sim, state.eim, state.has_r = nil, nil, false
8380         end
8381     else
8382         if d == 'an' or d == 'l' then
8383             if nodes[q][3] then -- nil except after an explicit dir
8384                 state.sim = item -- so we move sim 'inside' the group
8385             else
8386                 state.sim = state.sim or item
8387             end
8388             state.eim = item
8389         elseif d == 'r' and state.sim then
8390             head, state = insert_implicit(head, state, outer)
8391         elseif d == 'r' then
8392             state.sim, state.eim = nil, nil
8393         end
8394     end
8395
8396     if isdir then
8397         last = d          -- Don't search back - best save now
8398     elseif d == 'on' and state.san then
8399         state.san = state.san or item
8400         state.ean = item
8401     end
8402
8403 end
8404
8405 head = node.prev(head) or head
8406 % \end{macrocode}
8407 %
8408 % Now direction nodes has been distributed with relation to characters
8409 % and spaces, we need to take into account \TeX-specific elements in
8410 % the node list, to move them at an appropriate place. Firstly, with
8411 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8412 % that the latter are still discardable.
8413 %
8414 % \begin{macrocode}
8415 --- FIXES ---
8416 if has_hyperlink then
8417     local flag, linking = 0, 0
8418     for item in node.traverse(head) do

```



```

8419     if item.id == DIR then
8420         if item.dir == '+TRT' or item.dir == '+TLT' then
8421             flag = flag + 1
8422         elseif item.dir == '-TRT' or item.dir == '-TLT' then
8423             flag = flag - 1
8424         end
8425     elseif item.id == 8 and item.subtype == 19 then
8426         linking = flag
8427     elseif item.id == 8 and item.subtype == 20 then
8428         if linking > 0 then
8429             if item.prev.id == DIR and
8430                 (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8431                 d = node.new(DIR)
8432                 d.dir = item.prev.dir
8433                 node.remove(head, item.prev)
8434                 node.insert_after(head, item, d)
8435             end
8436         end
8437         linking = 0
8438     end
8439 end
8440 end
8441
8442 for item in node.traverse_id(10, head) do
8443     local p = item
8444     local flag = false
8445     while p.prev and p.prev.id == 14 do
8446         flag = true
8447         p = p.prev
8448     end
8449     if flag then
8450         node.insert_before(head, p, node.copy(item))
8451         node.remove(head, item)
8452     end
8453 end
8454
8455 return head
8456 end
8457
8457 function Babel.unset_atdir(head)
8458     local ATDIR = Babel.attr_dir
8459     for item in node.traverse(head) do
8460         node.set_attribute(item, ATDIR, 0x80)
8461     end
8462     return head
8463 end
8464 </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.

```
8465 ⟨*nil⟩
8466 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8467 \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.

```
8468 \ifx\l@nil\undefined
8469 \newlanguage\l@nil
8470 \@namedef{bbl@hyphendata@the\l@nil}{}{}% Remove warning
8471 \let\bbl@elt\relax
8472 \edef\bbl@languages{% Add it to the list of languages
8473 \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
8474 \fi
```

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

```
8475 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

`\captionnil`

`\datenil`

```
8476 \let\captionnil\@empty
8477 \let\datenil\@empty
```

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

```
8478 \def\bbl@inidata@nil{%
8479 \bbl@elt{identification}{tag.ini}{und}%
8480 \bbl@elt{identification}{load.level}{0}%
8481 \bbl@elt{identification}{charset}{utf8}%
8482 \bbl@elt{identification}{version}{1.0}%
8483 \bbl@elt{identification}{date}{2022-05-16}%
8484 \bbl@elt{identification}{name.local}{nil}%
8485 \bbl@elt{identification}{name.english}{nil}%
8486 \bbl@elt{identification}{name.babel}{nil}%
8487 \bbl@elt{identification}{tag.bcp47}{und}%
8488 \bbl@elt{identification}{language.tag.bcp47}{und}%
8489 \bbl@elt{identification}{tag.opentype}{dflt}%
8490 \bbl@elt{identification}{script.name}{Latin}%
8491 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8492 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8493 \bbl@elt{identification}{level}{1}%
8494 \bbl@elt{identification}{encodings}{}%
8495 \bbl@elt{identification}{derivate}{no}}
8496 \@namedef{bbl@tbcpl@nil}{und}
8497 \@namedef{bbl@lbcpl@nil}{und}
8498 \@namedef{bbl@casing@nil}{und}
8499 \@namedef{bbl@lotf@nil}{dflt}
8500 \@namedef{bbl@elname@nil}{nil}
8501 \@namedef{bbl@lname@nil}{nil}
8502 \@namedef{bbl@esname@nil}{Latin}
8503 \@namedef{bbl@sname@nil}{Latin}
8504 \@namedef{bbl@sbcpl@nil}{Latn}
8505 \@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.

```
8506 \ldf@finish{nil}
8507 ⟨/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.

```
8508 <<Compute Julian day>> ≡
8509 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8510 \def\bbl@cs@gregleap#1{%
8511   (\bbl@fpmo{#1}{4} == 0) &&
8512   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
8513 \def\bbl@cs@jd#1#2#3{% year, month, day
8514   \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8515     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8516     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8517     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8518 <</Compute Julian day>>
```

13.1. Islamic

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

```
8519 <ca-islamic>
8520 <@Compute Julian day>
8521 % == islamic (default)
8522 % Not yet implemented
8523 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}
```

The Civil calendar.

```
8524 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8525   ((#3 + ceil(29.5 * (#2 - 1)) +
8526     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8527     1948439.5) - 1) }
8528 \namedef\bbl@ca@islamic-civil++{\bbl@ca@islamicvl{x}{+2}}
8529 \namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl{x}{+1}}
8530 \namedef\bbl@ca@islamic-civil{\bbl@ca@islamicvl{x}{}}
8531 \namedef\bbl@ca@islamic-civil-{\bbl@ca@islamicvl{x}{-1}}
8532 \namedef\bbl@ca@islamic-civil--{\bbl@ca@islamicvl{x}{-2}}
8533 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8534   \edef\bbl@tempa{%
8535     \fpeval{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 } }%
8536   \edef#5{%
8537     \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) } }%
8538   \edef#6{\fpeval{
8539     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) } }%
8540   \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).

```
8541 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8542 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8543 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8544 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8545 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8546 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8547 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8548 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8549 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8550 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8551 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8552 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8553 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
```

```

8554 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8555 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8556 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8557 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8558 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8559 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8560 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8561 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8562 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8563 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8564 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8565 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8566 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8567 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8568 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8569 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8570 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8571 65401,65431,65460,65490,65520}
8572 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8573 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8574 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{0}}
8575 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8576   \ifnum#2>2014 \ifnum#2<2038
8577     \bbl@afterfi\expandafter\@gobble
8578   \fi\fi
8579   {\bbl@error{year-out-range}{2014-2038}}}%
8580 \edef\bbl@tempd{\fpeval{ % (Julian) day
8581   \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8582 \count@\@ne
8583 \bbl@foreach\bbl@cs@umalqura@data{%
8584   \advance\count@\@ne
8585   \ifnum##1>\bbl@tempd\else
8586     \edef\bbl@tempe{\the\count@}%
8587     \edef\bbl@tempb{##1}%
8588   \fi}%
8589 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8590 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1) / 12) }}% annus
8591 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8592 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8593 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8594 \bbl@add\bbl@precalendar{%
8595   \bbl@replace\bbl@ld@calendar{-civil}}}%
8596 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8597 \bbl@replace\bbl@ld@calendar{+}}}%
8598 \bbl@replace\bbl@ld@calendar{-}}}%
8599 </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`

```

8600 < *ca-hebrew>
8601 \newcount\bbl@cntcommon
8602 \def\bbl@remainder#1#2#3{%
8603   #3=#1\relax
8604   \divide #3 by #2\relax
8605   \multiply #3 by -#2\relax
8606   \advance #3 by #1\relax}%
8607 \newif\ifbbl@divisible
8608 \def\bbl@checkifdivisible#1#2{%
8609   {\countdef\tmp=0
8610     \bbl@remainder{#1}{#2}{\tmp}%

```

```

8611 \ifnum \tmp=0
8612 \global\bbl@divisibletrue
8613 \else
8614 \global\bbl@divisiblefalse
8615 \fi}}
8616 \newif\ifbbl@gregleap
8617 \def\bbl@ifgregleap#1{%
8618 \bbl@checkifdivisible{#1}{4}%
8619 \ifbbl@divisible
8620 \bbl@checkifdivisible{#1}{100}%
8621 \ifbbl@divisible
8622 \bbl@checkifdivisible{#1}{400}%
8623 \ifbbl@divisible
8624 \bbl@gregleaptrue
8625 \else
8626 \bbl@gregleapfalse
8627 \fi
8628 \else
8629 \bbl@gregleaptrue
8630 \fi
8631 \else
8632 \bbl@gregleapfalse
8633 \fi
8634 \ifbbl@gregleap}
8635 \def\bbl@gregdayspriormonths#1#2#3{%
8636 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8637 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8638 \bbl@ifgregleap{#2}%
8639 \ifnum #1 > 2
8640 \advance #3 by 1
8641 \fi
8642 \fi
8643 \global\bbl@cntcommon=#3}%
8644 #3=\bbl@cntcommon}
8645 \def\bbl@gregdaysprioryears#1#2{%
8646 {\countdef\tmpc=4
8647 \countdef\tmpb=2
8648 \tmpb=#1\relax
8649 \advance \tmpb by -1
8650 \tmpc=\tmpb
8651 \multiply \tmpc by 365
8652 #2=\tmpc
8653 \tmpc=\tmpb
8654 \divide \tmpc by 4
8655 \advance #2 by \tmpc
8656 \tmpc=\tmpb
8657 \divide \tmpc by 100
8658 \advance #2 by -\tmpc
8659 \tmpc=\tmpb
8660 \divide \tmpc by 400
8661 \advance #2 by \tmpc
8662 \global\bbl@cntcommon=#2\relax}%
8663 #2=\bbl@cntcommon}
8664 \def\bbl@absfromgreg#1#2#3#4{%
8665 {\countdef\tmpd=0
8666 #4=#1\relax
8667 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8668 \advance #4 by \tmpd
8669 \bbl@gregdaysprioryears{#3}{\tmpd}%
8670 \advance #4 by \tmpd
8671 \global\bbl@cntcommon=#4\relax}%
8672 #4=\bbl@cntcommon}
8673 \newif\ifbbl@hebrleap

```

```

8674 \def\bbl@checkleaphebrewyear#1{%
8675   {\countdef\tmpa=0
8676    \countdef\tmpb=1
8677    \tmpa=#1\relax
8678    \multiply\tmpa by 7
8679    \advance\tmpa by 1
8680    \bbl@remainder{\tmpa}{19}{\tmpb}%
8681    \ifnum\tmpb < 7
8682      \global\bbl@hebrleaptrue
8683    \else
8684      \global\bbl@hebrleapfalse
8685    \fi}}
8686 \def\bbl@hebreleapsedmonths#1#2{%
8687   {\countdef\tmpa=0
8688    \countdef\tmpb=1
8689    \countdef\tmpc=2
8690    \tmpa=#1\relax
8691    \advance\tmpa by -1
8692    #2=\tmpa
8693    \divide#2 by 19
8694    \multiply#2 by 235
8695    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8696    \tmpc=\tmpb
8697    \multiply\tmpb by 12
8698    \advance#2 by \tmpb
8699    \multiply\tmpc by 7
8700    \advance\tmpc by 1
8701    \divide\tmpc by 19
8702    \advance#2 by \tmpc
8703    \global\bbl@cntcommon=#2}%
8704   #2=\bbl@cntcommon}
8705 \def\bbl@hebreleapseddays#1#2{%
8706   {\countdef\tmpa=0
8707    \countdef\tmpb=1
8708    \countdef\tmpc=2
8709    \bbl@hebreleapsedmonths{#1}{#2}%
8710    \tmpa=#2\relax
8711    \multiply\tmpa by 13753
8712    \advance\tmpa by 5604
8713    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8714    \divide\tmpa by 25920
8715    \multiply#2 by 29
8716    \advance#2 by 1
8717    \advance#2 by \tmpa
8718    \bbl@remainder{#2}{7}{\tmpa}%
8719    \ifnum\tmpc < 19440
8720      \ifnum\tmpc < 9924
8721      \else
8722        \ifnum\tmpa=2
8723          \bbl@checkleaphebrewyear{#1}% of a common year
8724          \ifbbl@hebrleap
8725            \else
8726              \advance#2 by 1
8727            \fi
8728          \fi
8729        \ifnum\tmpc < 16789
8730        \else
8731          \ifnum\tmpa=1
8732            \advance#1 by -1
8733            \bbl@checkleaphebrewyear{#1}% at the end of leap year
8734            \ifbbl@hebrleap
8735              \advance#2 by 1

```

```

8737         \fi
8738     \fi
8739 \fi
8740 \else
8741     \advance #2 by 1
8742 \fi
8743 \bbl@remainder{#2}{7}{\tmpa}%
8744 \ifnum \tmpa=0
8745     \advance #2 by 1
8746 \else
8747     \ifnum \tmpa=3
8748         \advance #2 by 1
8749     \else
8750         \ifnum \tmpa=5
8751             \advance #2 by 1
8752         \fi
8753     \fi
8754 \fi
8755 \global\bbl@cntcommon=#2\relax}%
8756 #2=\bbl@cntcommon}
8757 \def\bbl@daysinhebrewyear#1#2{%
8758     {\countdef\tmpe=12
8759     \bbl@hebrelapseddays{#1}{\tmpe}%
8760     \advance #1 by 1
8761     \bbl@hebrelapseddays{#1}{#2}%
8762     \advance #2 by -\tmpe
8763     \global\bbl@cntcommon=#2}%
8764     #2=\bbl@cntcommon}
8765 \def\bbl@hebrdayspriormonths#1#2#3{%
8766     {\countdef\tmpf= 14
8767     #3=\ifcase #1
8768         0 \or
8769         0 \or
8770         30 \or
8771         59 \or
8772         89 \or
8773         118 \or
8774         148 \or
8775         148 \or
8776         177 \or
8777         207 \or
8778         236 \or
8779         266 \or
8780         295 \or
8781         325 \or
8782         400
8783     \fi
8784     \bbl@checkleaphebrewyear{#2}%
8785     \ifbbl@hebrleap
8786         \ifnum #1 > 6
8787             \advance #3 by 30
8788         \fi
8789     \fi
8790     \bbl@daysinhebrewyear{#2}{\tmpf}%
8791     \ifnum #1 > 3
8792         \ifnum \tmpf=353
8793             \advance #3 by -1
8794         \fi
8795         \ifnum \tmpf=383
8796             \advance #3 by -1
8797         \fi
8798     \fi
8799     \ifnum #1 > 2

```

```

8800     \ifnum \tmpf=355
8801         \advance #3 by 1
8802     \fi
8803     \ifnum \tmpf=385
8804         \advance #3 by 1
8805     \fi
8806 \fi
8807 \global\bbl@cntcommon=#3\relax}%
8808 #3=\bbl@cntcommon}
8809 \def\bbl@absfromhebr#1#2#3#4{%
8810     {#4=#1\relax
8811     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8812     \advance #4 by #1\relax
8813     \bbl@hebreleaseddays{#3}{#1}%
8814     \advance #4 by #1\relax
8815     \advance #4 by -1373429
8816     \global\bbl@cntcommon=#4\relax}%
8817 #4=\bbl@cntcommon}
8818 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8819     {\countdef\tmpx= 17
8820     \countdef\tmpy= 18
8821     \countdef\tmpz= 19
8822     #6=#3\relax
8823     \global\advance #6 by 3761
8824     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8825     \tmpz=1 \tmpy=1
8826     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8827     \ifnum \tmpx > #4\relax
8828         \global\advance #6 by -1
8829         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8830     \fi
8831     \advance #4 by -\tmpx
8832     \advance #4 by 1
8833     #5=#4\relax
8834     \divide #5 by 30
8835     \loop
8836         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8837         \ifnum \tmpx < #4\relax
8838             \advance #5 by 1
8839             \tmpy=\tmpx
8840         \repeat
8841     \global\advance #5 by -1
8842     \global\advance #4 by -\tmpy}}
8843 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8844 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8845 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8846     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8847     \bbl@hebrfromgreg
8848     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8849     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8850 \edef#4{\the\bbl@hebyear}%
8851 \edef#5{\the\bbl@hebrmonth}%
8852 \edef#6{\the\bbl@hebrday}}
8853 /ca-hebrew

```

13.3. Persian

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

```
8854 *ca-persian
```



```

8855 <@Compute Julian day@>
8856 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8857 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8858 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8859 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8860 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8861 \bbl@afterfi\expandafter\@gobble
8862 \fi\fi
8863 {\bbl@error{year-out-range}{2013-2050}{}}}%
8864 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8865 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8866 \edef\bbl@tempc{\fpeval{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8867 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8868 \ifnum\bbl@tempc<\bbl@tempb
8869 \edef\bbl@tempa{\fpeval{\bbl@tempa-1}}% go back 1 year and redo
8870 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8871 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8872 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8873 \fi
8874 \edef#4{\fpeval{\bbl@tempa-621}}% set Jalali year
8875 \edef#6{\fpeval{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8876 \edef#5{\fpeval{% set Jalali month
8877 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8878 \edef#6{\fpeval{% set Jalali day
8879 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}%
8880 </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.

```

8881 <*ca-coptic>
8882 <@Compute Julian day@>
8883 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8884 \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8885 \edef\bbl@tempc{\fpeval{\bbl@tempd - 1825029.5}}%
8886 \edef#4{\fpeval{%
8887 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8888 \edef\bbl@tempc{\fpeval{%
8889 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8890 \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
8891 \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8892 </ca-coptic>
8893 <*ca-ethiopic>
8894 <@Compute Julian day@>
8895 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8896 \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8897 \edef\bbl@tempc{\fpeval{\bbl@tempd - 1724220.5}}%
8898 \edef#4{\fpeval{%
8899 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8900 \edef\bbl@tempc{\fpeval{%
8901 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8902 \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
8903 \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8904 </ca-ethiopic>

```

13.5. Julian

Based on [ReinDersh].

```

8905 <*ca-julian>
8906 <@Compute Julian day@>
8907 \def\bbl@ca@julian#1-#2-#3\@@#4#5#6{%

```

```

8908 \edef\bbl@tempj{\fpeval{floor(\bbl@cs@jd{#1}{#2}{#3}) + .5}}%
8909 \edef\bbl@tempa{\fpeval{\bbl@tempj + 32082.5}}%
8910 \edef\bbl@tempb{\fpeval{floor((4 * \bbl@tempa + 3) / 1461)}}%
8911 \edef\bbl@tempc{\fpeval{\bbl@tempa - floor(1461*\bbl@tempb/4)}}%
8912 \edef\bbl@tempd{\fpeval{floor((5 * \bbl@tempc + 2) / 153)}}%
8913 \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
8914 \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
8915 \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}%
8916 </ca-julian>

```

13.6. Buddhist

That's very simple.

```

8917 <*ca-buddhist>
8918 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8919 \edef#4{\number\numexpr#1+543\relax}%
8920 \edef#5{#2}%
8921 \edef#6{#3}}
8922 </ca-buddhist>
8923 %
8924 % \subsection{Chinese}
8925 %
8926 % Brute force, with the Julian day of first day of each month. The
8927 % table has been computed with the help of \textsf{python-lunardate} by
8928 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8929 % is 2015-2044.
8930 %
8931 % \begin{macrocode}
8932 <*ca-chinese>
8933 \ExplSyntaxOn
8934 <@Compute Julian day@>
8935 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8936 \edef\bbl@tempd{\fpeval{%
8937 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8938 \count@\z@
8939 \@tempcnta=2015
8940 \bbl@foreach\bbl@cs@chinese@data{%
8941 \ifnum##1>\bbl@tempd\else
8942 \advance\count@\@ne
8943 \ifnum\count@>12
8944 \count@\@ne
8945 \advance\@tempcnta\@ne\fi
8946 \bbl@xin@{,##1,},{,\bbl@cs@chinese@leap,}%
8947 \ifin@
8948 \advance\count@\m@ne
8949 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8950 \else
8951 \edef\bbl@tempe{\the\count@}%
8952 \fi
8953 \edef\bbl@tempb{##1}%
8954 \fi}%
8955 \edef#4{\the\@tempcnta}%
8956 \edef#5{\bbl@tempe}%
8957 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8958 \def\bbl@cs@chinese@leap{%
8959 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8960 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8961 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8962 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8963 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8964 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8965 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8966 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%

```

```

8967 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8968 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8969 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8970 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8971 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8972 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8973 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8974 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8975 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8976 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8977 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8978 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8979 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8980 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8981 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8982 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8983 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8984 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8985 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8986 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8987 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8988 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8989 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8990 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8991 10896,10926,10956,10986,11015,11045,11074,11103}
8992 \ExplSyntaxOff
8993 </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 `localhyphen.tex` or whatever they like, but they mustn’t diddle with `hyphen.tex` (or `plain.tex` except to preload additional fonts).

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

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

```

8994 <{*bplain | blplain}
8995 \catcode`\{=1 % left brace is begin-group character
8996 \catcode`\}=2 % right brace is end-group character
8997 \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).

```

8998 \openin 0 hyphen.cfg
8999 \ifeof0
9000 \else
9001 \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.

```

9002 \def\input #1 {%

```

```

9003 \let\input\input
9004 \a hyphen.cfg
9005 \let\undefined\undefined
9006 }
9007 \fi
9008 </bplain | bplain>

```

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

```

9009 <bplain>\a plain.tex
9010 <bplain>\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.

```

9011 <bplain>\def\fmtname{babel-plain}
9012 <bplain>\def\fmtname{babel-lplain}

```

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

14.2. Emulating some \LaTeX features

The file `babel.def` expects some definitions made in the $\text{\LaTeX} 2_{\epsilon}$ 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).

```

9013 <<*Emulate LaTeX>> ≡
9014 \def\@empty{}
9015 \def\loadlocalcfg#1{%
9016   \openin0#1.cfg
9017   \ifeof0
9018     \closein0
9019   \else
9020     \closein0
9021     {\immediate\write16{*****}%
9022      \immediate\write16{* Local config file #1.cfg used}%
9023      \immediate\write16{*}%
9024     }
9025   \input #1.cfg\relax
9026 \fi
9027 \@endoflfd}

```

14.3. General tools

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

```

9028 \long\def\@firstofone#1{#1}
9029 \long\def\@firstoftwo#1#2{#1}
9030 \long\def\@secondoftwo#1#2{#2}
9031 \def\@nnil{\@nil}
9032 \def\@gobbletwo#1#2{}
9033 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
9034 \def\@star@or@long#1{%
9035   \@ifstar
9036   {\let\l@ngrel@x\relax#1}%
9037   {\let\l@ngrel@x\long#1}}
9038 \let\l@ngrel@x\relax
9039 \def\@car#1#2\@nil{#1}
9040 \def\@cdr#1#2\@nil{#2}
9041 \let\@typeset@protect\relax
9042 \let\protected@edef\edef
9043 \long\def\@gobble#1{}
9044 \edef\@backslashchar{\expandafter\@gobble\string\}

```

```

9045 \def\strip@prefix#1>{}
9046 \def\g@addto@macro#1#2{{%
9047   \toks@\expandafter{#1#2}%
9048   \xdef#1{\the\toks@}}
9049 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
9050 \def\@nameuse#1{\csname #1\endcsname}
9051 \def\@ifundefined#1{%
9052   \expandafter\ifx\csname#1\endcsname\relax
9053     \expandafter\@firstoftwo
9054   \else
9055     \expandafter\@secondoftwo
9056   \fi}
9057 \def\@expandtwoargs#1#2#3{%
9058   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9059 \def\zap@space#1 #2{%
9060   #1%
9061   \ifx#2\@empty\else\expandafter\zap@space\fi
9062   #2}
9063 \let\bbl@trace\@gobble
9064 \def\bbl@error#1{% Implicit #2#3#4
9065   \begingroup
9066     \catcode`\=0   \catcode`\==12 \catcode`\`=12
9067     \catcode`\^M=5 \catcode`\%=14
9068     \input errbabel.def
9069   \endgroup
9070   \bbl@error{#1}}
9071 \def\bbl@warning#1{%
9072   \begingroup
9073     \newlinechar=`^^J
9074     \def\{^^J(babel) }%
9075     \message{\{#1}%
9076   \endgroup}
9077 \let\bbl@infowarn\bbl@warning
9078 \def\bbl@info#1{%
9079   \begingroup
9080     \newlinechar=`^^J
9081     \def\{^^J}%
9082     \wlog{#1}%
9083   \endgroup}

```

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

```

9084 \ifx\@preamblecmds\undefined
9085   \def\@preamblecmds{}
9086 \fi
9087 \def\@onlypreamble#1{%
9088   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9089     \@preamblecmds\do#1}}
9090 \@onlypreamble\@onlypreamble

```

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

```

9091 \def\begindocument{%
9092   \@begindocumenthook
9093   \global\let\@begindocumenthook\undefined
9094   \def\do##1{\global\let##1\undefined}%
9095   \@preamblecmds
9096   \global\let\do\noexpand}
9097 \ifx\@begindocumenthook\undefined
9098   \def\@begindocumenthook{}
9099 \fi
9100 \@onlypreamble\@begindocumenthook
9101 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```
9102 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
9103 \@onlypreamble\AtEndOfPackage
9104 \def\@endofldf{}
9105 \@onlypreamble\@endofldf
9106 \let\bbl@afterlang\@empty
9107 \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.

```
9108 \catcode`\&=\z@
9109 \ifx&\if@filesw\@undefined
9110   \expandafter\let\csname if@filesw\expandafter\endcsname
9111     \csname iffalse\endcsname
9112 \fi
9113 \catcode`\&=4
```

Mimic \LaTeX 's commands to define control sequences.

```
9114 \def\newcommand{\@star@or@long\new@command}
9115 \def\new@command#1{%
9116   \@testopt{\@newcommand#1}0}
9117 \def\@newcommand#1[#2]{%
9118   \@ifnextchar [{\@xargdef#1[#2]}%
9119     {\@argdef#1[#2]}}
9120 \long\def\@argdef#1[#2]#3{%
9121   \@yargdef#1\@ne{#2}{#3}}
9122 \long\def\@xargdef#1[#2][#3]#4{%
9123   \expandafter\def\expandafter#1\expandafter{%
9124     \expandafter\@protected@testopt\expandafter #1%
9125     \csname\string#1\expandafter\endcsname{#3}}}%
9126 \expandafter\@yargdef \csname\string#1\endcsname
9127 \tw@{#2}{#4}}
9128 \long\def\@yargdef#1#2#3{%
9129   \@tempcnta#3\relax
9130   \advance \@tempcnta \@ne
9131   \let\@hash\@relax
9132   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9133   \@tempcntb #2%
9134   \@whilenum\@tempcntb <\@tempcnta
9135   \do{%
9136     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
9137     \advance\@tempcntb \@ne}%
9138   \let\@hash###%
9139   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9140 \def\providecommand{\@star@or@long\provide@command}
9141 \def\provide@command#1{%
9142   \begingroup
9143     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9144   \endgroup
9145   \expandafter\@ifundefined\@gtempa
9146     {\def\reserved@a{\new@command#1}}%
9147     {\let\reserved@a\relax
9148       \def\reserved@a{\new@command\reserved@a}}%
9149   \reserved@a}%
9150 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9151 \def\declare@robustcommand#1{%
9152   \edef\reserved@a{\string#1}%
9153   \def\reserved@b{#1}%
9154   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9155   \edef#1{%
9156     \ifx\reserved@a\reserved@b
```

```

9157      \noexpand\x@protect
9158      \noexpand#1%
9159      \fi
9160      \noexpand\protect
9161      \expandafter\noexpand\csname
9162      \expandafter@gobble\string#1 \endcsname
9163    }%
9164    \expandafter\new@command\csname
9165      \expandafter@gobble\string#1 \endcsname
9166  }
9167  \def\x@protect#1{%
9168    \ifx\protect@typeset@protect\else
9169      \@x@protect#1%
9170    \fi
9171  }
9172  \catcode`\&=\z@ % Trick to hide conditionals
9173  \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`.

```

9174  \def\bbl@tempa{\csname newif\endcsname&ifin@}
9175  \catcode`\&=4
9176  \ifx\in@\@undefined
9177    \def\in@#1#2{%
9178      \def\in@##1#1##2##3\in@{%
9179        \ifx\in@##2\in@false\else\in@true\fi}%
9180      \in@##2#1\in@\in@}
9181  \else
9182    \let\bbl@tempa\@empty
9183  \fi
9184  \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).

```

9185  \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.

```

9186  \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.

```

9187  \ifx\@tempcnta\@undefined
9188    \csname newcount\endcsname\@tempcnta\relax
9189  \fi
9190  \ifx\@tempcntb\@undefined
9191    \csname newcount\endcsname\@tempcntb\relax
9192  \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`).

```

9193  \ifx\bye\@undefined
9194    \advance\count10 by -2\relax
9195  \fi
9196  \ifx\@ifnextchar\@undefined
9197    \def\@ifnextchar#1#2#3{%
9198      \let\reserved@d=#1%
9199      \def\reserved@a{#2}\def\reserved@b{#3}%
9200      \futurelet\@let@token\@ifnch}

```

```

9201 \def\@ifnch{%
9202   \ifx\@let@token\@sptoken
9203     \let\reserved@c\@xifnch
9204   \else
9205     \ifx\@let@token\reserved@d
9206       \let\reserved@c\reserved@a
9207     \else
9208       \let\reserved@c\reserved@b
9209     \fi
9210   \fi
9211   \reserved@c}
9212 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
9213 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
9214 \fi
9215 \def\@testopt#1#2{%
9216   \ifnextchar[#{1}{#1[#2]}}
9217 \def\@protected@testopt#1{%
9218   \ifx\protect\@typeset@protect
9219     \expandafter\@testopt
9220   \else
9221     \@x@protect#1%
9222   \fi}
9223 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9224   #2\relax}\fi}
9225 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9226   \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9227 \def\DeclareTextCommand{%
9228   \@dec@text@cmd\providecommand
9229 }
9230 \def\ProvideTextCommand{%
9231   \@dec@text@cmd\providecommand
9232 }
9233 \def\DeclareTextSymbol#1#2#3{%
9234   \@dec@text@cmd\chardef#1{#2}#3\relax
9235 }
9236 \def\@dec@text@cmd#1#2#3{%
9237   \expandafter\def\expandafter#2%
9238     \expandafter{%
9239       \csname#3-cmd\expandafter\endcsname
9240       \expandafter#2%
9241       \csname#3\string#2\endcsname
9242     }%
9243 %   \let\@ifdefinable\@rc@ifdefinable
9244   \expandafter#1\csname#3\string#2\endcsname
9245 }
9246 \def\@current@cmd#1{%
9247   \ifx\protect\@typeset@protect\else
9248     \noexpand#1\expandafter\@gobble
9249   \fi
9250 }
9251 \def\@changed@cmd#1#2{%
9252   \ifx\protect\@typeset@protect
9253     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9254       \expandafter\ifx\csname ?\string#1\endcsname\relax
9255         \expandafter\def\csname ?\string#1\endcsname{%
9256           \@changed@x@err{#1}%
9257         }%
9258       \fi
9259     \global\expandafter\let

```



```

9260         \csname\cf@encoding \string#1\expandafter\endcsname
9261         \csname ?\string#1\endcsname
9262     \fi
9263     \csname\cf@encoding\string#1%
9264         \expandafter\endcsname
9265 \else
9266     \noexpand#1%
9267 \fi
9268 }
9269 \def\@changed@x@err#1{%
9270     \errhelp{Your command will be ignored, type <return> to proceed}%
9271     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9272 \def\DeclareTextCommandDefault#1{%
9273     \DeclareTextCommand#1?%
9274 }
9275 \def\ProvideTextCommandDefault#1{%
9276     \ProvideTextCommand#1?%
9277 }
9278 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9279 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9280 \def\DeclareTextAccent#1#2#3{%
9281     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9282 }
9283 \def\DeclareTextCompositeCommand#1#2#3#4{%
9284     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9285     \edef\reserved@b{\string##1}%
9286     \edef\reserved@c{%
9287         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9288     \ifx\reserved@b\reserved@c
9289         \expandafter\expandafter\expandafter\ifx
9290             \expandafter\@car\reserved@a\relax\relax\@nil
9291             \@text@composite
9292     \else
9293         \edef\reserved@b##1{%
9294             \def\expandafter\noexpand
9295                 \csname#2\string#1\endcsname###1{%
9296                 \noexpand\@text@composite
9297                 \expandafter\noexpand\csname#2\string#1\endcsname
9298                 ###1\noexpand\@empty\noexpand\@text@composite
9299                 {##1}%
9300             }%
9301         }%
9302         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9303     \fi
9304     \expandafter\def\csname\expandafter\string\csname
9305         #2\endcsname\string#1-\string#3\endcsname{#4}
9306 \else
9307     \errhelp{Your command will be ignored, type <return> to proceed}%
9308     \errmessage{\string\DeclareTextCompositeCommand\space used on
9309         inappropriate command \protect#1}
9310 \fi
9311 }
9312 \def\@text@composite#1#2#3\@text@composite{%
9313     \expandafter\@text@composite@x
9314         \csname\string#1-\string#2\endcsname
9315 }
9316 \def\@text@composite@x#1#2{%
9317     \ifx#1\relax
9318         #2%
9319     \else
9320         #1%
9321     \fi
9322 }

```

```

9323 %
9324 \def\@strip@args#1:#2-#3\@strip@args{#2}
9325 \def\DeclareTextComposite#1#2#3#4{%
9326   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9327   \bgroup
9328     \lccode`\@=#4%
9329     \lowercase{%
9330   \egroup
9331   \reserved@a @%
9332 }%
9333 }
9334 %
9335 \def\UseTextSymbol#1#2{#2}
9336 \def\UseTextAccent#1#2#3{}
9337 \def\@use@text@encoding#1{}
9338 \def\DeclareTextSymbolDefault#1#2{%
9339   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9340 }
9341 \def\DeclareTextAccentDefault#1#2{%
9342   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9343 }
9344 \def\cf@encoding{OT1}

```

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

```

9345 \DeclareTextAccent{"}{OT1}{127}
9346 \DeclareTextAccent{'}{OT1}{19}
9347 \DeclareTextAccent{^}{OT1}{94}
9348 \DeclareTextAccent{\`}{OT1}{18}
9349 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9350 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9351 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`}
9352 \DeclareTextSymbol{\textquoteleft}{OT1}{``}`
9353 \DeclareTextSymbol{\textquoteright}{OT1}{``'}`
9354 \DeclareTextSymbol{\i}{OT1}{16}
9355 \DeclareTextSymbol{\ss}{OT1}{25}

```

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

```

9356 \ifx\scriptsize\undefined
9357   \let\scriptsize\sevenrm
9358 \fi

```

And a few more “dummy” definitions.

```

9359 \def\language{english}%
9360 \let\bbl@opt@shorthands\@nnil
9361 \def\bbl@ifshorthand#1#2#3{#2}%
9362 \let\bbl@language@opts\@empty
9363 \let\bbl@provide@locale\relax
9364 \ifx\babeloptionstrings\undefined
9365   \let\bbl@opt@strings\@nnil
9366 \else
9367   \let\bbl@opt@strings\babeloptionstrings
9368 \fi
9369 \def\BabelStringsDefault{generic}
9370 \def\bbl@tempa{normal}
9371 \ifx\babeloptionmath\bbl@tempa
9372   \def\bbl@mathnormal{\noexpand\textormath}
9373 \fi
9374 \def\AfterBabelLanguage#1#2{}
9375 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9376 \let\bbl@afterlang\relax

```

```

9377 \def\bbl@opt@safe{BR}
9378 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
9379 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
9380 \expandafter\newif\csname ifbbl@single\endcsname
9381 \chardef\bbl@bidimode\z@
9382 <</Emulate LaTeX>>

A proxy file:

9383 <*\plain>
9384 \input babel.def
9385 </\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 \TeX styles*, *TUGboat* 10 (1989) #3, pp. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\TeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018.
- [10] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International \TeX is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).