

Babel

Code

Version 25.7
2025/04/14

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	9
3.4	key=value options and other general option	10
3.5	Post-process some options	11
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	39
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	50
4.15.3	Shorthands for quotation marks	51
4.15.4	Umlauts and tremas	52
4.16	Layout	53
4.17	Load engine specific macros	54
4.18	Creating and modifying languages	54
4.19	Main loop in ‘provide’	61
4.20	Processing keys in ini	66
4.21	French spacing (again)	71
4.22	Handle language system	72
4.23	Numerals	73
4.24	Casing	74
4.25	Getting info	75
4.26	BCP 47 related commands	76
5	Adjusting the Babel behavior	77
5.1	Cross referencing macros	79
5.2	Layout	82
5.3	Marks	82
5.4	Other packages	83
5.4.1	ifthen	83
5.4.2	varioref	84
5.4.3	hhline	85
5.5	Encoding and fonts	85
5.6	Basic bidi support	87
5.7	Local Language Configuration	90
5.8	Language options	90

6	The kernel of Babel	94
7	Error messages	94
8	Loading hyphenation patterns	98
9	luatex + xetex: common stuff	102
10	Hooks for XeTeX and LuaTeX	106
10.1	XeTeX	106
10.2	Support for interchar	108
10.3	Layout	110
10.4	8-bit TeX	111
10.5	LuaTeX	112
10.6	Southeast Asian scripts	118
10.7	CJK line breaking	120
10.8	Arabic justification	122
10.9	Common stuff	126
10.10	Automatic fonts and ids switching	126
10.11	Bidi	133
10.12	Layout	135
10.13	Lua: transforms	145
10.14	Lua: Auto bidi with basic and basic-r	155
11	Data for CJK	166
12	The ‘nil’ language	166
13	Calendars	167
13.1	Islamic	168
13.2	Hebrew	169
13.3	Persian	173
13.4	Coptic and Ethiopic	174
13.5	Buddhist	174
14	Support for Plain T_EX (plain.def)	176
14.1	Not renaming hyphen.tex	176
14.2	Emulating some L ^A T _E X features	177
14.3	General tools	177
14.4	Encoding related macros	181
15	Acknowledgements	183

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=25.7>>
2 <<date=2025/04/14>>
```

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

```

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

```

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

```

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

```

For each element in the comma separated `<key>=<value>` list, execute `<code>` with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the `<key>` alone, it passes `\@empty` (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@> %%NB%%
227   The multilingual framework for pdfLaTeX, LuaLaTeX and XeLaTeX]

```

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

```

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}

```

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

```

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

```

265 <@Basic macros@>
266 \ifpackagewith{babel}{silent}
267   {\let\bbl@info\@gobble
268    \let\bbl@infowarn\@gobble
269    \let\bbl@warning\@gobble}
270 {}
271 %
272 \def\AfterBabelLanguage#1{%
273   \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.

```

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

```

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

```

312 \bbl@trace{key=value and another general options}
313 \bbl@csarg\let\tempa\expandafter\csname opt@babel.sty\endcsname
314 \def\bbl@tempb#1.#2{% Remove trailing dot
315   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
316 \def\bbl@tempe#1=#2\@@{%
317   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
318 \def\bbl@tempd#1.#2\@nnil{%%^A TODO. Refactor lists?
319   \ifx\@empty#2%
320     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
321   \else
322     \in@{,provide=}{, #1}%
323     \ifin@
324       \edef\bbl@tempc{%
325         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
326     \else
327       \in@{ $modifiers$ }{ $ #1$ }%%^A TODO. Allow spaces.
328       \ifin@
329         \bbl@tempe#2\@@
330       \else
331         \in@{=}{ #1}%
332         \ifin@
333           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
334         \else
335           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
336           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
337         \fi
338       \fi
339     \fi
340   \fi}
341 \let\bbl@tempc\@empty
342 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
343 \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.

```

344 \DeclareOption{KeepShorthandsActive}{}
345 \DeclareOption{activeacute}{}
346 \DeclareOption{activegrave}{}
347 \DeclareOption{debug}{}
348 \DeclareOption{noconfigs}{}
349 \DeclareOption{showlanguages}{}
350 \DeclareOption{silent}{}
351 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
352 \chardef\bbl@iniflag\z@
353 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
354 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
355 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
356 % Don't use. Experimental. TODO.
357 \newif\ifbbl@single
358 \DeclareOption{selectors=off}{\bbl@singletrue}
359 <@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.

```

360 \let\bbl@opt@shorthands\@nnil
361 \let\bbl@opt@config\@nnil
362 \let\bbl@opt@main\@nnil
363 \let\bbl@opt@headfoot\@nnil
364 \let\bbl@opt@layout\@nnil
365 \let\bbl@opt@provide\@nnil

```

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

```

366 \def\bbl@tempa#1=#2\bbl@tempa{%
367   \bbl@csarg\ifx{opt@#1}\@nnil
368     \bbl@csarg\edef{opt@#1}{#2}%
369   \else
370     \bbl@error{bad-package-option}{#1}{#2}{}%
371   \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.

```

372 \let\bbl@language@opts\@empty
373 \DeclareOption*{%
374   \bbl@xin@{\string=}{\CurrentOption}%
375   \ifin@
376     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
377   \else
378     \bbl@add@list\bbl@language@opts{\CurrentOption}%
379   \fi}

```

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

```

380 \ProcessOptions*

```

3.5. Post-process some options

```

381 \ifx\bbl@opt@provide\@nnil
382   \let\bbl@opt@provide\@empty % %%% MOVE above
383 \else
384   \chardef\bbl@iniflag\@ne
385   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
386     \in@{,provide,},{, #1,}%
387     \ifin@
388       \def\bbl@opt@provide{#2}%
389     \fi}

```

390 \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=...

```
391 \bbl@trace{Conditional loading of shorthands}
392 \def\bbl@sh@string#1{%
393   \ifx#1\@empty\else
394     \ifx#1t\string~%
395     \else\ifx#1c\string,%
396     \else\string#1%
397     \fi\fi
398     \expandafter\bbl@sh@string
399   \fi}
400 \ifx\bbl@opt@shorthands\@nnil
401   \def\bbl@ifshorthand#1#2#3{#2}%
402 \else\ifx\bbl@opt@shorthands\@empty
403   \def\bbl@ifshorthand#1#2#3{#3}%
404 \else
```

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

```
405 \def\bbl@ifshorthand#1{%
406   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
407   \ifin@
408     \expandafter\@firstoftwo
409   \else
410     \expandafter\@secondoftwo
411   \fi}
```

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

```
412 \edef\bbl@opt@shorthands{%
413   \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.

```
414 \bbl@ifshorthand{'}%
415   {\PassOptionsToPackage{activeacute}{babel}}{}
416 \bbl@ifshorthand{`}%
417   {\PassOptionsToPackage{activegrave}{babel}}{}
418 \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.

```
419 \ifx\bbl@opt@headfoot\@nnil\else
420   \g@addto@macro\@resetactivechars{%
421     \set@typeset@protect
422     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
423     \let\protect\noexpand}
424 \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.

```
425 \ifx\bbl@opt@safe\@undefined
426   \def\bbl@opt@safe{BR}
427   % \let\bbl@opt@safe\@empty % Pending of \cite
428 \fi
```

For layout an auxiliary macro is provided, available for packages and language styles.

Optimization: if there is no layout, just do nothing.

```
429 \bbl@trace{Defining IfBabelLayout}
430 \ifx\bbl@opt@layout\@nnil
431   \newcommand\IfBabelLayout[3]{#3}%
432 \else
433   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
```

```

434 \in@{,layout,}{, #1,}%
435 \ifin@
436 \def\bbl@opt@layout{#2}%
437 \bbl@replace\bbl@opt@layout{ }{.}%
438 \fi}
439 \newcommand\IfBabelLayout[1]{%
440 \@expandtwoargs\in@{. #1.}{.\bbl@opt@layout.}%
441 \ifin@
442 \expandafter\@firstoftwo
443 \else
444 \expandafter\@secondoftwo
445 \fi}
446 \fi
447 </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.

```

448 < *core>
449 \ifx\ldf@quit\undefined\else
450 \endinput\fi % Same line!
451 <@Make sure ProvidesFile is defined@>
452 \ProvidesFile{babel.def}[<@date@> v<@version@> Babel common definitions]
453 \ifx\AtBeginDocument\undefined %^^A TODO. change test.
454 <@Emulate LaTeX@>
455 \fi
456 <@Basic macros@>
457 </core>

```

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

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

```

458 < *package | core>
459 \def\bbl@version{<@version@>}
460 \def\bbl@date{<@date@>}
461 <@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.

```

462 \def\adddialect#1#2{%
463 \global\chardef#1#2\relax
464 \bbl@usehooks{adddialect}{#1}{#2}}%
465 \begingroup
466 \count@#1\relax
467 \def\bbl@elt##1##2##3##4{%
468 \ifnum\count@=#2\relax
469 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
470 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
471 set to \expandafter\string\csname l@##1\endcsname\%
472 (\string\language\the\count@). Reported}%
473 \def\bbl@elt####1####2####3####4{%
474 \fi}%
475 \bbl@cs{languages}%
476 \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.

```

477 \def\bbl@fixname#1{%
478   \begingroup
479   \def\bbl@tempe{\l@}%
480   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
481   \bbl@tempd
482     {\lowercase\expandafter{\bbl@tempd}%
483      {\uppercase\expandafter{\bbl@tempd}%
484       \@empty
485        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
486         \uppercase\expandafter{\bbl@tempd}}}%
487       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
488        \lowercase\expandafter{\bbl@tempd}}}%
489   \@empty
490   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
491   \bbl@tempd
492   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}
493 \def\bbl@iflanguage#1{%
494   \@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` or it is `\relax`.

```

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

```

```

531 \ifx\bbl@bcp\relax
532 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
533 \fi
534 \fi\fi}
535 \let\bbl@initoload\relax

```

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

```

536 \def\iflanguage#1{%
537 \bbl@iflanguage{#1}{%
538 \ifnum\csname l@#1\endcsname=\language
539 \expandafter\@firstoftwo
540 \else
541 \expandafter\@secondoftwo
542 \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.

```

543 \let\bbl@select@type\z@
544 \edef\selectlanguage{%
545 \noexpand\protect
546 \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`.

```

547 \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.

```

548 \let\xstring\string

```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

\bbl@pop@language But when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's `aftergroup` mechanism to help us. The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

\bbl@language@stack The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```

549 \def\bbl@language@stack{}

```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

\bbl@push@language

\bbl@pop@language The stack is simply a list of languagenames, separated with a ‘+’ sign; the push function can be simple:

```

550 \def\bbl@push@language{%
551   \ifx\language\undefined\else
552     \ifx\currentgrouplevel\undefined
553       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
554     \else
555       \ifnum\currentgrouplevel=\z@
556         \xdef\bbl@language@stack{\language+}%
557       \else
558         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
559       \fi
560     \fi
561 \fi}

```

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

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

```

562 \def\bbl@pop@lang#1+#2\@@{%
563   \edef\language{#1}%
564   \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).

```

565 \let\bbl@ifrestoring\@secondoftwo
566 \def\bbl@pop@language{%
567   \expandafter\bbl@pop@lang\bbl@language@stack\@@
568   \let\bbl@ifrestoring\@firstoftwo
569   \expandafter\bbl@set@language\expandafter{\language}%
570   \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).

```

571 \chardef\localeid\z@
572 \gdef\bbl@id@last{0} % No real need for a new counter
573 \def\bbl@id@assign{%
574   \bbl@ifunset\bbl@id@\language}%
575   {\count@\bbl@id@last\relax
576   \advance\count@\@ne
577   \global\bbl@csarg\chardef{id@\language}\count@
578   \xdef\bbl@id@last{\the\count@}%
579   \ifcase\bbl@engine\or
580     \directlua{
581       Babel.locale_props[\bbl@id@last] = {}
582       Babel.locale_props[\bbl@id@last].name = '\language'
583       Babel.locale_props[\bbl@id@last].vars = {}
584     }%
585   \fi}%
586 {}%
587 \chardef\localeid\bbl@c{l{id@}}

```

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

```

588 \expandafter\def\csname selectlanguage \endcsname#1{%

```

```

589 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
590 \bbl@push@language
591 \aftergroup\bbl@pop@language
592 \bbl@set@language{#1}}
593 \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).

```

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

```

```
639 \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`.

```
640 \newif\ifbbl@usedategroup
641 \let\bbl@savextras\@empty
642 \def\bbl@switch#1{% from select@, foreign@
643 % restore
644 \originalTeX
645 \expandafter\def\expandafter\originalTeX\expandafter{%
646 \csname noextras#1\endcsname
647 \let\originalTeX\@empty
648 \babel@beginsave}%
649 \bbl@usehooks{afterreset}{}%
650 \languageshortands{none}%
651 % set the locale id
652 \bbl@id@assign
653 % switch captions, date
654 \bbl@bsphack
655 \ifcase\bbl@select@type
656 \csname captions#1\endcsname\relax
657 \csname date#1\endcsname\relax
658 \else
659 \bbl@xin@{,captions,}{,\bbl@select@opts,}%
660 \ifin@
661 \csname captions#1\endcsname\relax
662 \fi
663 \bbl@xin@{,date,}{,\bbl@select@opts,}%
664 \ifin@ % if \foreign... within \<language>date
665 \csname date#1\endcsname\relax
666 \fi
667 \fi
668 \bbl@esphack
669 % switch extras
670 \csname bbl@preextras@#1\endcsname
671 \bbl@usehooks{beforeextras}{}%
672 \csname extras#1\endcsname\relax
673 \bbl@usehooks{afterextras}{}%
674 % > babel-ensure
675 % > babel-sh-<short>
676 % > babel-bidi
677 % > babel-fontspec
678 \let\bbl@savextras\@empty
679 % hyphenation - case mapping
680 \ifcase\bbl@opt@hyphenmap\or
681 \def\BabelLower##1##2{\lccode##1=##2\relax}%
682 \ifnum\bbl@hymapsel>4\else
683 \csname\language @bbl@hyphenmap\endcsname
684 \fi
685 \chardef\bbl@opt@hyphenmap\z@
686 \else
```

```

687 \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
688 \csname\language @bbbl@hyphenmap\endcsname
689 \fi
690 \fi
691 \let\bbbl@hymapsel\@cclv
692 % hyphenation - select rules
693 \ifnum\csname l@\language\endcsname=\l@unhyphenated
694 \edef\bbbl@tempa{u}%
695 \else
696 \edef\bbbl@tempa{\bbbl@cl{\lnbrk}}%
697 \fi
698 % linebreaking - handle u, e, k (v in the future)
699 \bbbl@xin@{/u}{/\bbbl@tempa}%
700 \ifin@ \else \bbbl@xin@{/e}{/\bbbl@tempa} \fi % elongated forms
701 \ifin@ \else \bbbl@xin@{/k}{/\bbbl@tempa} \fi % only kashida
702 \ifin@ \else \bbbl@xin@{/p}{/\bbbl@tempa} \fi % padding (e.g., Tibetan)
703 \ifin@ \else \bbbl@xin@{/v}{/\bbbl@tempa} \fi % variable font
704 % hyphenation - save mins
705 \babel@savevariable\lefthyphenmin
706 \babel@savevariable\righthyphenmin
707 \ifnum\bbbl@engine=\@ne
708 \babel@savevariable\hyphenationmin
709 \fi
710 \ifin@
711 % unhyphenated/kashida/elongated/padding = allow stretching
712 \language\l@unhyphenated
713 \babel@savevariable\emergencystretch
714 \emergencystretch\maxdimen
715 \babel@savevariable\hbadness
716 \hbadness\@M
717 \else
718 % other = select patterns
719 \bbbl@patterns{#1}%
720 \fi
721 % hyphenation - set mins
722 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
723 \set@hyphenmins\tw@\thr@@\relax
724 \@nameuse{bbbl@hyphenmins@}%
725 \else
726 \expandafter\expandafter\expandafter\set@hyphenmins
727 \csname #1hyphenmins\endcsname\relax
728 \fi
729 \@nameuse{bbbl@hyphenmins@}%
730 \@nameuse{bbbl@hyphenmins@\language}%
731 \@nameuse{bbbl@hyphenatmin@}%
732 \@nameuse{bbbl@hyphenatmin@\language}%
733 \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.

```

734 \long\def\otherlanguage#1{%
735 \def\bbbl@selectortname{other}%
736 \ifnum\bbbl@hymapsel=\@cclv\let\bbbl@hymapsel\thr@@\fi
737 \csname selectlanguage \endcsname{#1}%
738 \ignorespaces}

```

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

```

739 \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.

```

740 \expandafter\def\csname otherlanguage*\endcsname{%
741   \ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s{}}
742 \def\bbl@otherlanguage@s[#1]#2{%
743   \def\bbl@selectorname{other*}%
744   \ifnum\bbl@hymapset=\@ccclv\chardef\bbl@hymapset4\relax\fi
745   \def\bbl@select@opts{#1}%
746   \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”.

```

747 \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.

```

748 \providecommand\bbl@beforeforeign{}
749 \edef\foreignlanguage{%
750   \noexpand\protect
751   \expandafter\noexpand\csname foreignlanguage \endcsname}
752 \expandafter\def\csname foreignlanguage \endcsname{%
753   \@ifstar\bbl@foreign@s\bbl@foreign@x}
754 \providecommand\bbl@foreign@x[3][]{%
755   \begingroup
756     \def\bbl@selectorname{foreign}%
757     \def\bbl@select@opts{#1}%
758     \let\BabelText\@firstofone
759     \bbl@beforeforeign
760     \foreign@language{#2}%
761     \bbl@usehooks{foreign}{}%
762     \BabelText{#3}% Now in horizontal mode!
763   \endgroup}
764 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@par
765   \begingroup
766     {\par}%
767     \def\bbl@selectorname{foreign*}%
768     \let\bbl@select@opts\@empty
769     \let\BabelText\@firstofone
770     \foreign@language{#1}%
771     \bbl@usehooks{foreign*}{}%
772     \bbl@dirparastext
773     \BabelText{#2}% Still in vertical mode!
774     {\par}%
775   \endgroup}
776 \providecommand\BabelWrapText[1]{%

```

```

777 \def\bbl@tempa{\def\BabelText####1}%
778 \expandafter\bbl@tempa\expandafter{\BabelText{#1}}

```

\foreign@language This macro does the work for \foreignlanguage and the other language* 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.

```

779 \def\foreign@language#1{%
780 % set name
781 \edef\language#1%
782 \ifbbl@usedategroup
783 \bbl@add\bbl@select@opts{,date,}%
784 \bbl@usedategroupfalse
785 \fi
786 \bbl@fixname\language
787 \let\localename\language
788 % TODO. name@map here?
789 \bbl@provide@locale
790 \bbl@iflanguage\language{%
791 \let\bbl@select@type\@ne
792 \expandafter\bbl@switch\expandafter{\language}}

```

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

```

793 \def\IfBabelSelectorTF#1{%
794 \bbl@xin@{\bbl@select@name,}{,\zap@space#1 \@empty,}%
795 \ifin@
796 \expandafter\@firstoftwo
797 \else
798 \expandafter\@secondoftwo
799 \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.

```

800 \let\bbl@hyphlist\@empty
801 \let\bbl@hyphenation@\relax
802 \let\bbl@pttnlist\@empty
803 \let\bbl@patterns@\relax
804 \let\bbl@hymapsel=\@cclv
805 \def\bbl@patterns#1{%
806 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
807 \csname l@#1\endcsname
808 \edef\bbl@tempa{#1}%
809 \else
810 \csname l@#1:f@encoding\endcsname
811 \edef\bbl@tempa{#1:f@encoding}%
812 \fi
813 \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}}%
814 % > luatex
815 \@ifundefined{bbl@hyphenation@}{% Can be \relax!
816 \begin{group}
817 \bbl@xin@{\, \number\language,}{,\bbl@hyphlist}%
818 \ifin@ \else
819 \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}}%
820 \hyphenation%
821 \bbl@hyphenation@
822 \@ifundefined{bbl@hyphenation@#1}%
823 \@empty

```

```

824         {\space\csname bbl@hyphenation@#1\endcsname}}%
825         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
826         \fi
827     \endgroup}}

```

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

```

828 \def\hyphenrules#1{%
829     \edef\bbl@tempf{#1}%
830     \bbl@fixname\bbl@tempf
831     \bbl@iflanguage\bbl@tempf{%
832         \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
833         \ifx\languageshorthands\undefined\else
834             \languageshorthands{none}%
835         \fi
836         \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
837             \set@hyphenmins\tw@\thr@\relax
838         \else
839             \expandafter\expandafter\expandafter\set@hyphenmins
840             \csname\bbl@tempf hyphenmins\endcsname\relax
841         \fi}}
842 \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.

```

843 \def\providehyphenmins#1#2{%
844     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
845         \@namedef{#1hyphenmins}{#2}%
846     \fi}

```

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

```

847 \def\set@hyphenmins#1#2{%
848     \lefthyphenmin#1\relax
849     \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.

```

850 \ifx\ProvidesFile\undefined
851     \def\ProvidesLanguage#1[#2 #3 #4]{%
852         \wlog{Language: #1 #4 #3 <#2>}%
853     }
854 \else
855     \def\ProvidesLanguage#1{%
856         \begingroup
857         \catcode`\ 10 %
858         \@makeother\/%
859         \@ifnextchar[%]
860             {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
861     \def\@provideslanguage#1[#2]{%
862         \wlog{Language: #1 #2}%
863         \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
864     \endgroup}
865 \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`.

```
866 \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.

```
867 \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’:

```
868 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}{}
869 \let\uselocale\setlocale
870 \let\locale\setlocale
871 \let\selectlocale\setlocale
872 \let\textlocale\setlocale
873 \let\textlanguage\setlocale
874 \let\languagegettext\setlocale
```

4.2. Errors

\@nolanerr

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

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

When the format knows about `\PackageError` it must be $\LaTeX 2_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

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


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

4.3. More on selection

\babelensure The user command just parses the optional argument and creates a new macro named `\bb@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 `\bb@e@<language>` contains `\bb@ensure{<include>}{<exclude>}{<fontenc>}`, which in turn loops over the macros names in `\bb@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.

```

906 \bb@trace{Defining babelensure}
907 \newcommand\babelensure[2][]{%
908   \AddBabelHook{babel-ensure}{afterextras}{%
909     \ifcase\bb@select@type
910       \bb@cl{e}%
911     \fi}%
912   \begingroup
913     \let\bb@ens@include\@empty
914     \let\bb@ens@exclude\@empty
915     \def\bb@ens@fontenc{\relax}%
916     \def\bb@tempb##1{%
917       \ifx\@empty##1\else\noexpand##1\expandafter\bb@tempb\fi}%
918     \edef\bb@tempa{\bb@tempb#1\@empty}%
919     \def\bb@tempb##1=##2\@{\@namedef{\bb@ens@##1}{##2}}%
920     \bb@foreach\bb@tempa{\bb@tempb##1\@}%
921     \def\bb@tempc{\bb@ensure}%
922     \expandafter\bb@add\expandafter\bb@tempc\expandafter{%
923       \expandafter{\bb@ens@include}}%
924     \expandafter\bb@add\expandafter\bb@tempc\expandafter{%
925       \expandafter{\bb@ens@exclude}}%
926     \toks@ \expandafter{\bb@tempc}%
927     \bb@exp{%
928   \endgroup
929   \def<\bb@e@#2>{\the\toks@{\bb@ens@fontenc}}}%
930 \def\bb@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
931   \def\bb@tempb##1{% elt for (excluding) \bb@captionslist list
932     \ifx##1\undefined % 3.32 - Don't assume the macro exists
933       \edef##1{\noexpand\bb@nocaption
934         {\bb@stripslash##1}{\language\bb@stripslash##1}}%
935     \fi
936     \ifx##1\@empty\else
937       \in@{##1}{#2}%
938       \ifin@ \else
939         \bb@ifunset{\bb@ensure@\language\language}%
940         {\bb@exp{%
941           \\DeclareRobustCommand\<\bb@ensure@\language\language>[1]{%
942             \\foreignlanguage{\language\language}%
943             {\ifx\relax#3\else
944               \\fontencoding{#3}\\selectfont
945             \fi
946             #####1}}}%
947         }%
948         \toks@ \expandafter{##1}%
949         \edef##1{%
950           \bb@csarg\noexpand{\bb@ensure@\language\language}%
951           {\the\toks@}}%
952       \fi

```

```

953     \expandafter\bbbl@tempb
954     \fi}%
955 \expandafter\bbbl@tempb\bbbl@captionslist\today\@empty
956 \def\bbbl@tempa##1{% elt for include list
957     \ifx##1\@empty\else
958         \bbbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
959         \ifin@else
960             \bbbl@tempb##1\@empty
961         \fi
962     \expandafter\bbbl@tempa
963     \fi}%
964 \bbbl@tempa#1\@empty}
965 \def\bbbl@captionslist{%
966 \prefacename\refname\abstractname\bibname\chaptername\appendixname
967 \contentsname\listfigurename\listtablename\indexname\figurename
968 \tablename\partname\enclname\ccname\headtoname\pagename\seename
969 \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.

```

970 \bbbl@trace{Short tags}
971 \newcommand\babeltags[1]{%
972     \edef\bbbl@tempa{\zap@space#1 \@empty}%
973     \def\bbbl@tempb##1=##2\@{
974         \edef\bbbl@tempc{%
975             \noexpand\newcommand
976             \expandafter\noexpand\csname ##1\endcsname{%
977                 \noexpand\protect
978                 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
979             \noexpand\newcommand
980             \expandafter\noexpand\csname text##1\endcsname{%
981                 \noexpand\foreignlanguage{##2}}
982         \bbbl@tempc}%
983     \bbbl@for\bbbl@tempa\bbbl@tempa{%
984         \expandafter\bbbl@tempb\bbbl@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.

```

985 \bbbl@trace{Compatibility with language.def}
986 \ifx\directlua\@undefined\else
987     \ifx\bbbl@luapatterns\@undefined
988         \input luababel.def
989     \fi
990 \fi
991 \ifx\bbbl@languages\@undefined
992     \ifx\directlua\@undefined
993         \openin1 = language.def % TODO. Remove hardcoded number
994         \ifeof1
995             \closein1
996             \message{I couldn't find the file language.def}
997         \else
998             \closein1
999             \begingroup
1000             \def\addlanguage#1#2#3#4#5{%
1001                 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1002                     \global\expandafter\let\csname l@#1\endcsname\expandafter\endcsname
1003                     \csname lang@#1\endcsname
1004             \fi}%

```

```

1005      \def\uselanguage#1{%
1006      \input language.def
1007      \endgroup
1008      \fi
1009      \fi
1010      \chardef\l@english\z@
1011 \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.

```

1012 \def\addto#1#2{%
1013   \ifx#1\@undefined
1014     \def#1{#2}%
1015   \else
1016     \ifx#1\relax
1017       \def#1{#2}%
1018     \else
1019       {\toks@\expandafter{#1#2}%
1020        \xdef#1{\the\toks@}}%
1021     \fi
1022   \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.

```

1023 \bbl@trace{Hooks}
1024 \newcommand\AddBabelHook[3][[]]{%
1025   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1026   \def\bbl@tempa#1,#3=#2,##3\@empty{\def\bbl@tempb{##3}}%
1027   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1028   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1029     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1030     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1031   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1032 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1033 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1034 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1035 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1036   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1037   \def\bbl@elth##1{%
1038     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1039     \bbl@cs{ev@#2@#3}%
1040     \ifx\language\@undefined\else % Test required for Plain (?)
1041       \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1042       \def\bbl@elth##1{%
1043         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1#3}}%
1044         \bbl@cs{ev@#2@#1}%
1045       \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).

```

1046 \def\bbl@evargs{,% <- don't delete this comma
1047   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1048   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1049   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1050   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%

```

```

1051 beforestart=0, languagename=2, begindocument=1}
1052 \ifx\NewHook\@undefined\else % Test for Plain (?)
1053 \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1054 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1055 \fi

```

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

```

1056 \providecommand\PassOptionsToLocale[2]{%
1057 \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 @-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.

```

1058 \bbl@trace{Macros for setting language files up}
1059 \def\bbl@ldfinit{%
1060 \let\bbl@screset\@empty
1061 \let\BabelStrings\bbl@opt@string
1062 \let\BabelOptions\@empty
1063 \let\BabelLanguages\relax
1064 \ifx\originalTeX\@undefined
1065 \let\originalTeX\@empty
1066 \else
1067 \originalTeX
1068 \fi}
1069 \def\LdfInit#1#2{%
1070 \chardef\atcatcode=\catcode`\@
1071 \catcode`\@=11\relax
1072 \chardef\eqcatcode=\catcode`\=
1073 \catcode`\==12\relax
1074 \ifpackagewith{babel}{ensureinfo=off}}}%
1075 {\ifx\InputIfFileExists\@undefined\else
1076 \bbl@ifunset\bbl@lname@#1}%
1077 {{\let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
1078 \def\languagename{#1}%
1079 \bbl@id@assign
1080 \bbl@load@info{#1}}}%
1081 }}%
1082 \fi}%
1083 \expandafter\if\expandafter\@backslashchar
1084 \expandafter\@car\string#2\@nil
1085 \ifx#2\@undefined\else
1086 \ldf@quit{#1}%
1087 \fi
1088 \else
1089 \expandafter\ifx\csname#2\endcsname\relax\else
1090 \ldf@quit{#1}%

```

```

1091 \fi
1092 \fi
1093 \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.

```

1094 \def\ldf@quit#1{%
1095 \expandafter\main@language\expandafter{#1}%
1096 \catcode`\@=\atcatcode \let\atcatcode\relax
1097 \catcode`\==\eqcatcode \let\eqcatcode\relax
1098 \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.

```

1099 \def\bbl@afterldf{%
1100 \bbl@afterlang
1101 \let\bbl@afterlang\relax
1102 \let\BabelModifiers\relax
1103 \let\bbl@screset\relax}%
1104 \def\ldf@finish#1{%
1105 \loadlocalcfg{#1}%
1106 \bbl@afterldf
1107 \expandafter\main@language\expandafter{#1}%
1108 \catcode`\@=\atcatcode \let\atcatcode\relax
1109 \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 .

```

1110 \@onlypreamble\LdfInit
1111 \@onlypreamble\ldf@quit
1112 \@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.

```

1113 \def\main@language#1{%
1114 \def\bbl@main@language{#1}%
1115 \let\language\name\bbl@main@language
1116 \let\localename\bbl@main@language
1117 \let\mainlocalename\bbl@main@language
1118 \bbl@id@assign
1119 \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.

```

1120 \def\bbl@beforestart{%
1121 \def\@nolanerr##1{%
1122 \bbl@carg\chardef{l@##1}\z@
1123 \bbl@warning{Undefined language '##1' in aux.\Reported}}%
1124 \bbl@usehooks{beforestart}}}%
1125 \global\let\bbl@beforestart\relax}
1126 \AtBeginDocument{%
1127 {\@nameuse\bbl@beforestart}}% Group!
1128 \if@filesw
1129 \providecommand\babel@aux[2]{}%

```

```

1130 \immediate\write\@mainaux{\unexpanded{%
1131 \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1132 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1133 \fi
1134 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1135 \ifbbl@single % must go after the line above.
1136 \renewcommand\selectlanguage[1]{}%
1137 \renewcommand\foreignlanguage[2]{#2}%
1138 \global\let\babel@aux\@gobbletwo % Also as flag
1139 \fi}
1140 %
1141 \ifcase\bbl@engine\or
1142 \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1143 \fi

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

1144 \def\select@language@x#1{%
1145 \ifcase\bbl@select@type
1146 \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1147 \else
1148 \select@language{#1}%
1149 \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.

```

1150 \bbl@trace{Shorhands}
1151 \def\bbl@withactive#1#2{%
1152 \begingroup
1153 \lccode`~=#2\relax
1154 \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.

```

1155 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1156 \bbl@add\dospecials{do#1}% test @sanitize = \relax, for back. compat.
1157 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\makeother#1}}%
1158 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1159 \begingroup
1160 \catcode`#1\active
1161 \nfss@catcodes
1162 \ifnum\catcode`#1=\active
1163 \endgroup
1164 \bbl@add\nfss@catcodes{\makeother#1}%
1165 \else
1166 \endgroup
1167 \fi
1168 \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).

```
1169 \def\bbl@active@def#1#2#3#4{%
1170   \@namedef{#3#1}{%
1171     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1172       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1173     \else
1174       \bbl@afterfi\csname#2@sh@#1\endcsname
1175     \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.

```
1176   \long\@namedef{#3@arg#1}##1{%
1177     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1178       \bbl@afterelse\csname#4#1\endcsname##1%
1179     \else
1180       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1181     \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.

```
1182 \def\initiate@active@char#1{%
1183   \bbl@ifunset{active@char\string#1}%
1184   {\bbl@withactive
1185     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1186   {}}
```

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).

```
1187 \def\@initiate@active@char#1#2#3{%
1188   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1189   \ifx#1\@undefined
1190     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1191   \else
1192     \bbl@csarg\let{oridef@#2}#1%
1193     \bbl@csarg\edef{oridef@#2}{%
1194       \let\noexpand#1%
1195       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1196   \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*").

```
1197   \ifx#1#3\relax
1198     \expandafter\let\csname normal@char#2\endcsname#3%
1199   \else
1200     \bbl@info{Making #2 an active character}%
1201     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1202     \@namedef{normal@char#2}{%
1203       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
```

```

1204 \else
1205 \namedef{normal@char#2}{#3}%
1206 \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).

```

1207 \bbl@restoreactive{#2}%
1208 \AtBeginDocument{%
1209 \catcode`#2\active
1210 \if@files
1211 \immediate\write\@mainaux{\catcode`\string#2\active}%
1212 \fi}%
1213 \expandafter\bbl@add@special\csname#2\endcsname
1214 \catcode`#2\active
1215 \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}`).

```

1216 \let\bbl@tempa\@firstoftwo
1217 \if\string^#2%
1218 \def\bbl@tempa{\noexpand\textormath}%
1219 \else
1220 \ifx\bbl@mathnormal\@undefined\else
1221 \let\bbl@tempa\bbl@mathnormal
1222 \fi
1223 \fi
1224 \expandafter\edef\csname active@char#2\endcsname{%
1225 \bbl@tempa
1226 {\noexpand\if@safe@actives
1227 \noexpand\expandafter
1228 \expandafter\noexpand\csname normal@char#2\endcsname
1229 \noexpand\else
1230 \noexpand\expandafter
1231 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1232 \noexpand\fi}%
1233 {\expandafter\noexpand\csname normal@char#2\endcsname}}}%
1234 \bbl@csarg\edef{doactive#2}{%
1235 \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

$$\backslash active@prefix \langle char \rangle \backslash normal@char \langle char \rangle$$

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

```

1236 \bbl@csarg\edef{active@#2}{%
1237 \noexpand\active@prefix\noexpand#1%
1238 \expandafter\noexpand\csname active@char#2\endcsname}%
1239 \bbl@csarg\edef{normal@#2}{%
1240 \noexpand\active@prefix\noexpand#1%
1241 \expandafter\noexpand\csname normal@char#2\endcsname}%
1242 \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.

```

1243 \bbl@active@def#2\user@group{user@active}{language@active}%
1244 \bbl@active@def#2\language@group{language@active}{system@active}%
1245 \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 T_EX would see `\protect'\protect'`. To prevent this from happening a couple of shorthand needs to be defined at user level.

```
1246 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1247   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1248 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1249   {\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.

```
1250 \if\string'#2%
1251   \let\prim@s\bbl@prim@s
1252   \let\active@math@prime#1%
1253 \fi
1254 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1255 << *More package options >> ≡
1256 \DeclareOption{math=active}{}
1257 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1258 << /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*.

```
1259 \ifpackagewith{babel}{KeepShorthandsActive}%
1260   {\let\bbl@restoreactive\@gobble}%
1261   {\def\bbl@restoreactive#1{%
1262     \bbl@exp{%
1263       \\\AfterBabelLanguage\\CurrentOption
1264       {\catcode`#1=\the\catcode`#1\relax}%
1265       \\\AtEndOfPackage
1266       {\catcode`#1=\the\catcode`#1\relax}}}%
1267   \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.

```
1268 \def\bbl@sh@select#1#2{%
1269   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1270     \bbl@afterelse\bbl@scndcs
1271   \else
1272     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1273   \fi}
```

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

```
1274 \begingroup
1275 \bbl@ifunset{ifincsname}%^^A Ugly. Correct? Only Plain?
1276   {\gdef\active@prefix#1{%
1277     \ifx\protect\@typeset@protect
```

```

1278     \else
1279     \ifx\protect\@unexpandable@protect
1280     \noexpand#1%
1281     \else
1282     \protect#1%
1283     \fi
1284     \expandafter\@gobble
1285     \fi}}
1286 {\gdef\active@prefix#1{%
1287   \ifincsname
1288   \string#1%
1289   \expandafter\@gobble
1290   \else
1291   \ifx\protect\@typeset@protect
1292   \else
1293   \ifx\protect\@unexpandable@protect
1294   \noexpand#1%
1295   \else
1296   \protect#1%
1297   \fi
1298   \expandafter\expandafter\expandafter\@gobble
1299   \fi
1300   \fi}}
1301 \endgroup

```

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

```

1302 \newif\if@safe@actives
1303 \@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.

```

1304 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

\bbl@activate

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

```

1305 \chardef\bbl@activated\z@
1306 \def\bbl@activate#1{%
1307   \chardef\bbl@activated\@ne
1308   \bbl@withactive{\expandafter\let\expandafter}#1%
1309   \csname bbl@active@\string#1\endcsname}
1310 \def\bbl@deactivate#1{%
1311   \chardef\bbl@activated\tw@
1312   \bbl@withactive{\expandafter\let\expandafter}#1%
1313   \csname bbl@normal@\string#1\endcsname}

```

\bbl@firstcs

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

```

1314 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1315 \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.

```
1316 \def\babel@texpdf#1#2#3#4{%
1317   \ifx\texorpdfstring\undefined
1318     \textormath{#1}{#3}%
1319   \else
1320     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1321   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1322   \fi}
1323 %
1324 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1325 \def\@decl@short#1#2#3\@nil#4{%
1326   \def\bbl@tempa{#3}%
1327   \ifx\bbl@tempa\@empty
1328     \expandafter\let\csname #1@sh@\string#2\sel\endcsname\bbl@scndcs
1329     \bbl@ifunset{#1@sh@\string#2@}{}%
1330     {\def\bbl@tempa{#4}%
1331      \expandafter\ifx\csname#1@sh@\string#2\endcsname\bbl@tempa
1332      \else
1333        \bbl@info
1334          {Redefining #1 shorthand \string#2\\%
1335           in language \CurrentOption}%
1336      \fi}%
1337     \@namedef{#1@sh@\string#2@}{#4}%
1338   \else
1339     \expandafter\let\csname #1@sh@\string#2\sel\endcsname\bbl@firstcs
1340     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1341     {\def\bbl@tempa{#4}%
1342      \expandafter\ifx\csname#1@sh@\string#2@\string#3\endcsname\bbl@tempa
1343      \else
1344        \bbl@info
1345          {Redefining #1 shorthand \string#2\string#3\\%
1346           in language \CurrentOption}%
1347      \fi}%
1348     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1349   \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.

```
1350 \def\textormath{%
1351   \ifmmode
1352     \expandafter\@secondoftwo
1353   \else
1354     \expandafter\@firstoftwo
1355   \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’.

```
1356 \def\user@group{user}
1357 \def\language@group{english} %^^A I don't like defaults
1358 \def\system@group{system}
```

\useshortands 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 shortcuts, so a starred version is also provided which activates them always after the language has been switched.

```

1359 \def\useshortands{%
1360   \@ifstar\bb\@usesesh@s{\bb\@usesesh@x{}}
1361 \def\bb\@usesesh@s#1{%
1362   \bb\@usesesh@x
1363   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb\@activate{#1}}}%
1364   {#1}}
1365 \def\bb\@usesesh@x#1#2{%
1366   \bb\@ifshorthand{#2}%
1367   {\def\user@group{user}%
1368    \initiate@active@char{#2}%
1369    #1%
1370    \bb\@activate{#2}}%
1371   {\bb\@error{shorthand-is-off}{#2}{}}}

```

\defineshorthand Currently we only support two groups of user level shortcuts, named internally user and user@(*language*) (language-dependent user shortcuts). 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 \bb\@set@user@generic); we make also sure {} and \protect are taken into account in this new top level.

```

1372 \def\user@language@group{user@\language@group}
1373 \def\bb\@set@user@generic#1#2{%
1374   \bb\@ifunset{user@generic@active#1}%
1375   {\bb\@active@def#1\user@language@group{user@active}{user@generic@active}%
1376    \bb\@active@def#1\user@group{user@generic@active}{\language@active}%
1377    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1378     \expandafter\noexpand\csname normal@char#1\endcsname}%
1379    \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1380     \expandafter\noexpand\csname user@active#1\endcsname}}%
1381   \@empty}
1382 \newcommand\defineshorthand[3][user]{%
1383   \edef\bb\@tempa{\zap@space#1 \@empty}%
1384   \bb\@for\bb\@tempb\bb\@tempa{%
1385     \if*\expandafter\@car\bb\@tempb\@nil
1386     \edef\bb\@tempb{user@\expandafter\@gobble\bb\@tempb}%
1387     \@expandtwoargs
1388     \bb\@set@user@generic{\expandafter\string\@car#2\@nil}\bb\@tempb
1389   \fi
1390   \declare@shorthand{\bb\@tempb}{#2}{#3}}

```

\languageshortands A user level command to change the language from which shortcuts 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.

```

1391 \def\languageshortands#1{%
1392   \bb\@ifsamestring{none}{#1}{}}%
1393   \bb\@once{short-\localename-#1}{%
1394     \bb\@info{'\localename' activates '#1' shortcuts.\@Reported }}}%
1395   \def\language@group{#1}}

```

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

```

1396 \def\aliasshorthand#1#2{%
1397   \bb\@ifshorthand{#2}%
1398   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1399     \ifx\document\@notprerr
1400     \@notshorthand{#2}%
1401     \else
1402     \initiate@active@char{#2}%

```

```

1403      \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1404      \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1405      \bbl@activate{#2}%
1406      \fi
1407      \fi}%
1408      {\bbl@error{shorthand-is-off}}{#2}{}}

```

\@notshorthand

```

1409 \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.

```

1410 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1411 \DeclareRobustCommand*\shorthandoff{%
1412   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1413 \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.

```

1414 \def\bbl@switch@sh#1#2{%
1415   \ifx#2\@nnil\else
1416     \bbl@ifunset{bbl@active@\string#2}%
1417     {\bbl@error{not-a-shorthand-b}}{#2}{}}%
1418     {\ifcase#1    off, on, off*
1419       \catcode`#2\relax
1420       \or
1421       \catcode`#2\active
1422       \bbl@ifunset{bbl@shdef@\string#2}%
1423       {}%
1424       {\bbl@withactive{\expandafter\let\expandafter}#2%
1425         \csname bbl@shdef@\string#2\endcsname
1426         \bbl@csarg\let{shdef@\string#2}\relax}%
1427       \ifcase\bbl@activated\or
1428         \bbl@activate{#2}%
1429       \else
1430         \bbl@deactivate{#2}%
1431       \fi
1432       \or
1433       \bbl@ifunset{bbl@shdef@\string#2}%
1434       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1435       {}%
1436       \csname bbl@oricat@\string#2\endcsname
1437       \csname bbl@oridef@\string#2\endcsname
1438       \fi}%
1439   \bbl@afterfi\bbl@switch@sh#1%
1440   \fi}

```

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

```

1441 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1442 \def\bbl@putsh#1{%
1443   \bbl@ifunset{bbl@active@\string#1}%
1444   {\bbl@putsh@i#1\@empty\@nnil}%
1445   {\csname bbl@active@\string#1\endcsname}}

```

```

1446 \def\bbl@putsh@i#1#2\@nnil{%
1447   \csname\language@group @sh@\string#1@%
1448     \ifx\@empty#2\else\string#2@\fi\endcsname}
1449 %
1450 \ifx\bbl@opt@shorthands\@nnil\else
1451   \let\bbl@s@initiate@active@char\initiate@active@char
1452   \def\initiate@active@char#1{%
1453     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1454   \let\bbl@s@switch@sh\bbl@switch@sh
1455   \def\bbl@switch@sh#1#2{%
1456     \ifx#2\@nnil\else
1457       \bbl@afterfi
1458       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1459     \fi}
1460   \let\bbl@s@activate\bbl@activate
1461   \def\bbl@activate#1{%
1462     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1463   \let\bbl@s@deactivate\bbl@deactivate
1464   \def\bbl@deactivate#1{%
1465     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1466 \fi

```

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

```

1467 \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.

```

1468 \def\bbl@prim@s{%
1469   \prime\futurelet\@let@token\bbl@pr@m@s}
1470 \def\bbl@if@primes#1#2{%
1471   \ifx#1\@let@token
1472     \expandafter\@firstoftwo
1473   \else\ifx#2\@let@token
1474     \bbl@afterelse\expandafter\@firstoftwo
1475   \else
1476     \bbl@afterfi\expandafter\@secondoftwo
1477   \fi\fi}
1478 \begingroup
1479   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1480   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1481   \lowercase{%
1482     \gdef\bbl@pr@m@s{%
1483       \bbl@if@primes" '%
1484         \pr@@@s
1485         {\bbl@if@primes*\^{\pr@@@t\egroup}}}
1486 \endgroup

```

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

```

1487 \initiate@active@char{~}
1488 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1489 \bbl@activate{~}

```

\OT1dpos

\Tldqpos 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.

```
1490 \expandafter\def\csname OTldqpos\endcsname{127}
1491 \expandafter\def\csname Tldqpos\endcsname{4}
```

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

```
1492 \ifx\f@encoding\undefined
1493   \def\f@encoding{OT1}
1494 \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.

```
1495 \bbl@trace{Language attributes}
1496 \newcommand\languageattribute[2]{%
1497   \def\bbl@tempc{#1}%
1498   \bbl@fixname\bbl@tempc
1499   \bbl@iflanguage\bbl@tempc{%
1500     \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.

```
1501     \ifx\bbl@known@attrs\undefined
1502       \in@false
1503     \else
1504       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1505     \fi
1506     \ifin@
1507       \bbl@warning{%
1508         You have more than once selected the attribute '##1'\%
1509         for language #1. Reported}%
1510     \else
```

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

```
1511       \bbl@exp{%
1512         \\ \bbl@add@list\\ \bbl@known@attrs{\bbl@tempc-##1}}%
1513       \edef\bbl@tempa{\bbl@tempc-##1}%
1514       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1515       {\csname\bbl@tempc @attr##1\endcsname}%
1516       {\@attrerr{\bbl@tempc}{##1}}%
1517     \fi}}
1518 \@onlypreamble\languageattribute
```

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

```
1519 \newcommand*{\@attrerr}[2]{%
1520   \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}`.

```
1521 \def\bbl@declare@ttribute#1#2#3{%
1522   \bbl@xin@{,#2,}{,\BabelModifiers,}%
```

```

1523 \ifin@
1524   \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1525 \fi
1526 \bbl@add@list\bbl@attributes{#1-#2}%
1527 \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret T_EX 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.

```

1528 \def\bbl@ifattributeset#1#2#3#4{%
1529   \ifx\bbl@known@attrs\@undefined
1530     \in@false
1531   \else
1532     \bbl@xin@{, #1-#2, }{, \bbl@known@attrs,}%
1533   \fi
1534   \ifin@
1535     \bbl@afterelse#3%
1536   \else
1537     \bbl@afterfi#4%
1538   \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 T_EX-code to be executed when the attribute is known and the T_EX-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.

```

1539 \def\bbl@ifknown@ttrib#1#2{%
1540   \let\bbl@tempa\@secondoftwo
1541   \bbl@loopx\bbl@tempb{#2}{%
1542     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1543   \ifin@
1544     \let\bbl@tempa\@firstoftwo
1545   \else
1546     \fi}%
1547   \bbl@tempa}

```

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

```

1548 \def\bbl@clear@ttribs{%
1549   \ifx\bbl@attributes\@undefined\else
1550     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1551       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1552     \let\bbl@attributes\@undefined
1553   \fi}
1554 \def\bbl@clear@ttrib#1-#2.{%
1555   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1556 \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.

```
1557 \bbl@trace{Macros for saving definitions}
1558 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1559 \newcount\babel@savecnt
1560 \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.

```
1561 \def\babel@save#1{%
1562   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1563   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1564     \expandafter{\expandafter,\bbl@savedextras,}}%
1565   \expandafter\in@\bbl@tempa
1566   \ifin@%else
1567     \bbl@add\bbl@savedextras{,{#1,}}%
1568     \bbl@carg\let\babel@number\babel@savecnt#1\relax
1569     \toks@{\expandafter{\originalTeX\let#1=}}%
1570     \bbl@exp{%
1571       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1572     \advance\babel@savecnt@ne
1573   \fi}
1574 \def\babel@savevariable#1{%
1575   \toks@{\expandafter{\originalTeX #1=}}%
1576   \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`.

```
1577 \def\bbl@redefine#1{%
1578   \edef\bbl@tempa{\bbl@stripslash#1}%
1579   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1580   \expandafter\def\csname\bbl@tempa\endcsname}
1581 \@onlypreamble\bbl@redefine
```

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

```
1582 \def\bbl@redefine@long#1{%
1583   \edef\bbl@tempa{\bbl@stripslash#1}%
1584   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1585   \long\expandafter\def\csname\bbl@tempa\endcsname}
1586 \@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_`.

```
1587 \def\bbl@redefineroobust#1{%
1588   \edef\bbl@tempa{\bbl@stripslash#1}%
1589   \bbl@ifunset{\bbl@tempa\space}%
1590   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
```

```

1591 \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1592 {\bbl@exp{\let\<org@\bbl@tempa\<\bbl@tempa\space>}}}%
1593 \@namedef{\bbl@tempa\space}}
1594 \@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.

```

1595 \def\bbl@frenchspacing{%
1596   \ifnum\the\sfcodes`.\!=\@m
1597     \let\bbl@nonfrenchspacing\relax
1598   \else
1599     \frenchspacing
1600     \let\bbl@nonfrenchspacing\nonfrenchspacing
1601   \fi}
1602 \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.

```

1603 \let\bbl@elt\relax
1604 \edef\bbl@fs@chars{%
1605   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1606   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1607   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1608 \def\bbl@pre@fs{%
1609   \def\bbl@elt##1##2##3{\sfcodes`##1=\the\sfcodes`##1\relax}%
1610   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1611 \def\bbl@post@fs{%
1612   \bbl@save@sfcodes
1613   \edef\bbl@tempa{\bbl@cl{frspc}}}%
1614 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1615 \if u\bbl@tempa % do nothing
1616 \else\if n\bbl@tempa % non french
1617   \def\bbl@elt##1##2##3{%
1618     \ifnum\sfcodes`##1=##2\relax
1619       \babel@savevariable{\sfcodes`##1}%
1620       \sfcodes`##1=##3\relax
1621     \fi}%
1622   \bbl@fs@chars
1623 \else\if y\bbl@tempa % french
1624   \def\bbl@elt##1##2##3{%
1625     \ifnum\sfcodes`##1=##3\relax
1626       \babel@savevariable{\sfcodes`##1}%
1627       \sfcodes`##1=##2\relax
1628     \fi}%
1629   \bbl@fs@chars
1630 \fi\fi\fi}

```

4.12. Hyphens

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

```

1631 \bbl@trace{Hyphens}
1632 \@onlypreamble\babelhyphenation
1633 \AtEndOfPackage{%
1634   \newcommand\babelhyphenation[2][\@empty]{%
1635     \ifx\bbl@hyphenation@\relax

```

```

1636 \let\bbl@hyphenation@\@empty
1637 \fi
1638 \ifx\bbl@hyphlist\@empty\else
1639 \bbl@warning{%
1640 You must not intermingle \string\selectlanguage\space and\\%
1641 \string\babelhyphenation\space or some exceptions will not\\%
1642 be taken into account. Reported}%
1643 \fi
1644 \ifx\@empty#1%
1645 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1646 \else
1647 \bbl@vforeach{#1}{%
1648 \def\bbl@tempa{##1}%
1649 \bbl@fixname\bbl@tempa
1650 \bbl@iflanguage\bbl@tempa{%
1651 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1652 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1653 }%
1654 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1655 #2}}}%
1656 \fi}}

```

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

```

1657 \ifx\NewDocumentCommand\@undefined\else
1658 \NewDocumentCommand\babelhyphenmins{sommo}{%
1659 \IfNoValueTF{#2}%
1660 {\protected@edef\bbl@hyphenmins@{\set@hyphenmins{#3}{#4}}}%
1661 \IfValueT{#5}{%
1662 \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1663 \IfBooleanT{#1}{%
1664 \leftthyphenmin=#3\relax
1665 \rightthyphenmin=#4\relax
1666 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1667 {\edef\bbl@tempb{\zap@space#2 \@empty}%
1668 \bbl@for\bbl@tempa\bbl@tempb{%
1669 \@namedef{bbl@hyphenmins@bbl@tempa}{\set@hyphenmins{#3}{#4}}}%
1670 \IfValueT{#5}{%
1671 \@namedef{bbl@hyphenatmin@bbl@tempa}{\hyphenationmin=#5\relax}}}%
1672 \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}}}}
1673 \fi

```

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

```

1674 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\zap@skip\fi}
1675 \def\bbl@t@one{T1}
1676 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}

```

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

```

1677 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1678 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1679 \def\bbl@hyphen{%
1680 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1681 \def\bbl@hyphen@i#1#2{%
1682 \lowercase{\bbl@ifunset{bbl@hy@#1#2\@empty}}%
1683 {\csname bbl@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1684 {\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.

```
1685 \def\bbl@usehyphen#1{%
1686   \leavevmode
1687   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1688   \nobreak\hskip\z@skip}
1689 \def\bbl@usehyphen#1{%
1690   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1691 \def\bbl@hyphenchar{%
1692   \ifnum\hyphenchar\font=\m@ne
1693     \babe\nullhyphen
1694   \else
1695     \char\hyphenchar\font
1696   \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.

```
1697 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1698 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1699 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1700 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1701 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1702 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1703 \def\bbl@hy@repeat{%
1704   \bbl@usehyphen{%
1705     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1706 \def\bbl@hy@@repeat{%
1707   \bbl@usehyphen{%
1708     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1709 \def\bbl@hy@empty{\hskip\z@skip}
1710 \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.

```
1711 \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.

```
1712 \bbl@trace{Multiencoding strings}
1713 \def\bbl@tglobal#1{\global\let#1#1}
```

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

```
1714 << *More package options >> ≡
1715 \DeclareOption{nocase}{}
1716 << /More package options >>
```

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

```
1717 << *More package options >> ≡
1718 \let\bbl@opt@strings\@nnil % accept strings=value
1719 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1720 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1721 \def\BabelStringsDefault{generic}
1722 << /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.

```

1723 \@onlypreamble\StartBabelCommands
1724 \def\StartBabelCommands{%
1725   \begingroup
1726   \@tempcnta="7F
1727   \def\bbl@tempa{%
1728     \ifnum\@tempcnta>"FF\else
1729       \catcode\@tempcnta=11
1730       \advance\@tempcnta\@ne
1731       \expandafter\bbl@tempa
1732     \fi}%
1733   \bbl@tempa
1734   <@Macros local to BabelCommands@>
1735   \def\bbl@provstring##1##2{%
1736     \providecommand##1{##2}%
1737     \bbl@tglobal##1}%
1738   \global\let\bbl@scafter\@empty
1739   \let\StartBabelCommands\bbl@startcmds
1740   \ifx\BabelLanguages\relax
1741     \let\BabelLanguages\CurrentOption
1742   \fi
1743   \begingroup
1744   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1745   \StartBabelCommands}
1746 \def\bbl@startcmds{%
1747   \ifx\bbl@screset\@nnil\else
1748     \bbl@usehooks{stopcommands}{}%
1749   \fi
1750   \endgroup
1751   \begingroup
1752   \@ifstar
1753     {\ifx\bbl@opt@strings\@nnil
1754       \let\bbl@opt@strings\BabelStringsDefault
1755     \fi
1756     \bbl@startcmds@i}%
1757   \bbl@startcmds@i}
1758 \def\bbl@startcmds@i##1##2{%
1759   \edef\bbl@L{\zap@space#1 \@empty}%
1760   \edef\bbl@G{\zap@space#2 \@empty}%
1761   \bbl@startcmds@ii}
1762 \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.

```

1763 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1764   \let\SetString\@gobbletwo
1765   \let\bbl@stringdef\@gobbletwo
1766   \let\AfterBabelCommands\@gobble
1767   \ifx\@empty#1%
1768     \def\bbl@sc@label{generic}%
1769     \def\bbl@encstring##1##2{%
1770       \ProvideTextCommandDefault##1{##2}%
1771       \bbl@tglobal##1%
1772       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1773 \let\bbl@sctest\in@true
1774 \else
1775 \let\bbl@sc@charset\space % <- zapped below
1776 \let\bbl@sc@fontenc\space % <- " "
1777 \def\bbl@tempa##1=##2\@nil{%
1778 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1779 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1780 \def\bbl@tempa##1 ##2{% space -> comma
1781 ##1%
1782 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1783 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1784 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1785 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1786 \def\bbl@encstring##1##2{%
1787 \bbl@foreach\bbl@sc@fontenc{%
1788 \bbl@ifunset{T@####1}%
1789 {}%
1790 {\ProvideTextCommand##1{####1}{##2}%
1791 \bbl@tglobal##1%
1792 \expandafter
1793 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1794 \def\bbl@sctest{%
1795 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}%
1796 \fi
1797 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1798 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1799 \let\AfterBabelCommands\bbl@aftercmds
1800 \let\SetString\bbl@setstring
1801 \let\bbl@stringdef\bbl@encstring
1802 \else % i.e., strings=value
1803 \bbl@sctest
1804 \ifin@
1805 \let\AfterBabelCommands\bbl@aftercmds
1806 \let\SetString\bbl@setstring
1807 \let\bbl@stringdef\bbl@provstring
1808 \fi\fi\fi
1809 \bbl@scswitch
1810 \ifx\bbl@G\@empty
1811 \def\SetString##1##2{%
1812 \bbl@error{missing-group}{##1}{}}}%
1813 \fi
1814 \ifx\@empty#1%
1815 \bbl@usehooks{defaultcommands}{}%
1816 \else
1817 \@expandtwoargs
1818 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1819 \fi}

```

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

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

```

1820 \def\bbl@forlang#1#2{%
1821 \bbl@for#1\bbl@L{%
1822 \bbl@xin@{,##1,}{,\BabelLanguages,}%
1823 \ifin@#2\relax\fi}}
1824 \def\bbl@scswitch{%
1825 \bbl@forlang\bbl@tempa{%
1826 \ifx\bbl@G\@empty\else

```

```

1827 \ifx\SetString@gobbletwo\else
1828 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1829 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1830 \ifin@else
1831 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1832 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1833 \fi
1834 \fi
1835 \fi}}
1836 \AtEndOfPackage{%
1837 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1838 \let\bbl@scswitch\relax}
1839 \@onlypreamble\EndBabelCommands
1840 \def\EndBabelCommands{%
1841 \bbl@usehooks{stopcommands}{}}%
1842 \endgroup
1843 \endgroup
1844 \bbl@scafter}
1845 \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.

```

1846 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1847 \bbl@forlang\bbl@tempa{%
1848 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1849 \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1850 {\bbl@exp{%
1851 \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1852 }%
1853 \def\BabelString{#2}%
1854 \bbl@usehooks{stringprocess}{}}%
1855 \expandafter\bbl@stringdef
1856 \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.

```

1857 \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.

```

1858 << *Macros local to BabelCommands >> ≡
1859 \def\SetStringLoop##1##2{%
1860 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1861 \count@\z@
1862 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1863 \advance\count@\@ne
1864 \toks@\expandafter{\bbl@tempa}%
1865 \bbl@exp{%
1866 \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1867 \count@=\the\count@\relax}}}%
1868 <</Macros local to BabelCommands >>

```

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

```

1869 \def\bbl@aftercmds#1{%
1870 \toks@\expandafter{\bbl@scafter#1}%
1871 \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.

```

1872 <<*Macros local to BabelCommands>> ≡
1873   \newcommand\SetCase[3][]{%
1874     \def\bbl@tempa####1####2{%
1875       \ifx####1\@empty\else
1876         \bbl@carg\bbl@add{extras\CurrentOption}{%
1877           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1878           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1879           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1880           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1881         \expandafter\bbl@tempa
1882       \fi}%
1883   \bbl@tempa##1\@empty\@empty
1884   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1885 <</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.

```

1886 <<*Macros local to BabelCommands>> ≡
1887   \newcommand\SetHyphenMap[1]{%
1888     \bbl@forlang\bbl@tempa{%
1889       \expandafter\bbl@stringdef
1890       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1891 <</Macros local to BabelCommands>>

```

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

```

1892 \newcommand\BabelLower[2]{% one to one.
1893   \ifnum\lccode#1=#2\else
1894     \babel@savevariable{\lccode#1}%
1895     \lccode#1=#2\relax
1896   \fi}
1897 \newcommand\BabelLowerMM[4]{% many-to-many
1898   \@tempcnta=#1\relax
1899   \@tempcntb=#4\relax
1900   \def\bbl@tempa{%
1901     \ifnum\@tempcnta>#2\else
1902       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1903       \advance\@tempcnta#3\relax
1904       \advance\@tempcntb#3\relax
1905       \expandafter\bbl@tempa
1906     \fi}%
1907   \bbl@tempa}
1908 \newcommand\BabelLowerM0[4]{% many-to-one
1909   \@tempcnta=#1\relax
1910   \def\bbl@tempa{%
1911     \ifnum\@tempcnta>#2\else
1912       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1913       \advance\@tempcnta#3
1914       \expandafter\bbl@tempa
1915     \fi}%
1916   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1917 <<*More package options>> ≡
1918 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1919 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1920 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1921 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1922 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1923 <</More package options>>

```


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

```

1924 \AtEndOfPackage{%
1925   \ifx\bbbl@opt@hyphenmap\@undefined
1926     \bbbl@xin@{,}\bbbl@language@opts}%
1927     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1928   \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.

```

1929 \newcommand\setlocalecaption{%^A Catch typos.
1930   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1931 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1932   \bbbl@trim@def\bbbl@tempa{#2}%
1933   \bbbl@xin@{.template}\bbbl@tempa}%
1934   \ifin@
1935     \bbbl@ini@captions@template{#3}{#1}%
1936   \else
1937     \edef\bbbl@tempd{%
1938       \expandafter\expandafter\expandafter
1939       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1940     \bbbl@xin@
1941       {\expandafter\string\csname #2name\endcsname}%
1942       {\bbbl@tempd}%
1943     \ifin@ % Renew caption
1944       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
1945     \ifin@
1946       \bbbl@exp{%
1947         \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1948         {\\bbbl@scset\<#2name>\<#1#2name>}%
1949         {}}%
1950     \else % Old way converts to new way
1951       \bbbl@ifunset{#1#2name}%
1952       {\bbbl@exp{%
1953         \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1954         \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1955         {\def\<#2name>\<#1#2name>}}%
1956         {}}}%
1957     {}%
1958   \fi
1959 \else
1960   \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
1961   \ifin@ % New way
1962     \bbbl@exp{%
1963       \\bbbl@add\<captions#1>{\\bbbl@scset\<#2name>\<#1#2name>}}%
1964       \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1965       {\\bbbl@scset\<#2name>\<#1#2name>}}%
1966       {}}%
1967     \else % Old way, but defined in the new way
1968       \bbbl@exp{%
1969         \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1970         \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1971         {\def\<#2name>\<#1#2name>}}%
1972         {}}%
1973     \fi%
1974   \fi
1975   \@namedef{#1#2name}{#3}%
1976   \toks@ \expandafter\bbbl@captionslist}%
1977   \bbbl@exp{\in@{\<#2name>}\the\toks@}%
1978   \ifin@ \else
1979     \bbbl@exp{\\bbbl@add\\bbbl@captionslist{\<#2name>}}%

```

```

1980      \bbl@toglobal\bbl@captionslist
1981      \fi
1982      \fi}
1983 %^^A \def\bbl@setcaption@s#1#2#3{} % Not yet implemented (w/o 'name')

```

4.15. Making glyphs available

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

```

1984 \bbl@trace{Macros related to glyphs}
1985 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1986      \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1987      \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.

```

1988 \def\save@sf@q#1{\leavevmode
1989      \begingroup
1990      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1991      \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.

```

1992 \ProvideTextCommand{\quotedblbase}{OT1}{%
1993      \save@sf@q{\set@low@box{\textquotedblright\}}%
1994      \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.

```

1995 \ProvideTextCommandDefault{\quotedblbase}{%
1996      \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1997 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1998      \save@sf@q{\set@low@box{\textquoteright\}}%
1999      \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.

```

2000 \ProvideTextCommandDefault{\quotesinglbase}{%
2001      \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

2002 \ProvideTextCommand{\guillemetleft}{OT1}{%
2003      \ifmmode
2004          \ll
2005      \else
2006          \save@sf@q{\nobreak
2007              \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2008          \fi}
2009 \ProvideTextCommand{\guillemetright}{OT1}{%
2010      \ifmmode
2011          \gg
2012      \else
2013          \save@sf@q{\nobreak

```

```

2014      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2015 \fi}
2016 \ProvideTextCommand{\guillemotleft}{OT1}{%
2017   \ifmmode
2018     \ll
2019   \else
2020     \save@sf@q{\nobreak
2021       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2022   \fi}
2023 \ProvideTextCommand{\guillemotright}{OT1}{%
2024   \ifmmode
2025     \gg
2026   \else
2027     \save@sf@q{\nobreak
2028       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2029   \fi}

```

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

```

2030 \ProvideTextCommandDefault{\guillemetleft}{%
2031   \UseTextSymbol{OT1}{\guillemetleft}}
2032 \ProvideTextCommandDefault{\guillemetright}{%
2033   \UseTextSymbol{OT1}{\guillemetright}}
2034 \ProvideTextCommandDefault{\guillemotleft}{%
2035   \UseTextSymbol{OT1}{\guillemotleft}}
2036 \ProvideTextCommandDefault{\guillemotright}{%
2037   \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2038 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2039   \ifmmode
2040     <%
2041   \else
2042     \save@sf@q{\nobreak
2043       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2044   \fi}
2045 \ProvideTextCommand{\guilsinglright}{OT1}{%
2046   \ifmmode
2047     >%
2048   \else
2049     \save@sf@q{\nobreak
2050       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2051   \fi}

```

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

```

2052 \ProvideTextCommandDefault{\guilsinglleft}{%
2053   \UseTextSymbol{OT1}{\guilsinglleft}}
2054 \ProvideTextCommandDefault{\guilsinglright}{%
2055   \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.

```

2056 \DeclareTextCommand{\ij}{OT1}{%
2057   i\kern-0.02em\bbl@allowhyphens j}
2058 \DeclareTextCommand{\IJ}{OT1}{%
2059   I\kern-0.02em\bbl@allowhyphens J}
2060 \DeclareTextCommand{\ij}{T1}{\char188}
2061 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2062 \ProvideTextCommandDefault{\ij}{%
2063   \UseTextSymbol{OT1}{\ij}}
2064 \ProvideTextCommandDefault{\IJ}{%
2065   \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).

```
2066 \def\crrtic@{\hrule height0.1ex width0.3em}
2067 \def\crttic@{\hrule height0.1ex width0.33em}
2068 \def\ddj@{%
2069   \setbox0\hbox{d}\dimen@=\ht0
2070   \advance\dimen@lex
2071   \dimen@.45\dimen@
2072   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2073   \advance\dimen@ii.5ex
2074   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2075 \def\DDJ@{%
2076   \setbox0\hbox{D}\dimen@=.55\ht0
2077   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2078   \advance\dimen@ii.15ex % correction for the dash position
2079   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2080   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2081   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2082 %
2083 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2084 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2085 \ProvideTextCommandDefault{\dj}{%
2086   \UseTextSymbol{OT1}{\dj}}
2087 \ProvideTextCommandDefault{\DJ}{%
2088   \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.

```
2089 \DeclareTextCommand{\SS}{OT1}{SS}
2090 \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.

```
2091 \ProvideTextCommandDefault{\glq}{%
2092   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2093 \ProvideTextCommand{\grq}{T1}{%
2094   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2095 \ProvideTextCommand{\grq}{TU}{%
2096   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2097 \ProvideTextCommand{\grq}{OT1}{%
2098   \save@sf@q{\kern-.0125em
2099     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%

```

```

2100 \kern.07em\relax}}
2101 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2102 \ProvideTextCommandDefault{\glqq}{%
2103 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2104 \ProvideTextCommand{\grqq}{T1}{%
2105 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2106 \ProvideTextCommand{\grqq}{TU}{%
2107 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2108 \ProvideTextCommand{\grqq}{0T1}{%
2109 \save@sf@q{\kern-.07em
2110 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2111 \kern.07em\relax}}
2112 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2113 \ProvideTextCommandDefault{\flq}{%
2114 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2115 \ProvideTextCommandDefault{\frq}{%
2116 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```

2117 \ProvideTextCommandDefault{\flqq}{%
2118 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2119 \ProvideTextCommandDefault{\frqq}{%
2120 \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).

```

2121 \def\umlauthigh{%
2122 \def\bbl@umlauta##1{\leavevmode\bgroup%
2123 \accent\csname\f@encoding dqpos\endcsname
2124 ##1\bbl@allowhyphens\egroup}%
2125 \let\bbl@umlaute\bbl@umlauta}
2126 \def\umlautlow{%
2127 \def\bbl@umlauta{\protect\lower@umlaut}}
2128 \def\umlautelow{%
2129 \def\bbl@umlaute{\protect\lower@umlaut}}
2130 \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.

```
2131 \expandafter\ifx\csname U@D\endcsname\relax
2132   \csname newdimen\endcsname\U@D
2133 \fi
```

The following code fools T_EX's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

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

```
2134 \def\lower@umlaut#1{%
2135   \leavevmode\bgroup
2136     \U@D lex%
2137     {\setbox\z@\hbox{%
2138       \char\csname f@encoding dqpos\endcsname}%
2139       \dimen@ -.45ex\advance\dimen@ \ht\z@
2140       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2141     \accent\csname f@encoding dqpos\endcsname
2142     \fontdimen5\font\U@D #1%
2143   \egroup}
```

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

```
2144 \AtBeginDocument{%
2145   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2146   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2147   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2148   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
2149   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2150   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2151   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2152   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2153   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2154   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2155   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}}
```

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

```
2156 \ifx\l@english\@undefined
2157   \chardef\l@english\z@
2158 \fi
2159 % The following is used to cancel rules in ini files (see Amharic).
2160 \ifx\l@unhyphenated\@undefined
2161   \newlanguage\l@unhyphenated
2162 \fi
```

4.16. Layout

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

```
2163 \bbl@trace{Bidi layout}
2164 \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.

```
2165 \bbl@trace{Input engine specific macros}
2166 \ifcase\bbl@engine
2167   \input txtbabel.def
2168 \or
2169   \input luababel.def
2170 \or
2171   \input xebabel.def
2172 \fi
2173 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2174 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2175 \ifx\babelposthyphenation\undefined
2176   \let\babelposthyphenation\babelprehyphenation
2177   \let\babelpatterns\babelprehyphenation
2178   \let\babelcharproperty\babelprehyphenation
2179 \fi
2180 \end{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.

```
2181 \begin{package}
2182 \bbl@trace{Creating languages and reading ini files}
2183 \let\bbl@extend@ini@gobble
2184 \newcommand\babelprovide[2][]{%
2185   \let\bbl@savelangname\languagename
2186   \edef\bbl@savelocaleid{\the\localeid}%
2187   % Set name and locale id
2188   \edef\languagename{#2}%
2189   \bbl@id@assign
2190   % Initialize keys
2191   \bbl@vforeach{captions,date,import,main,script,language,%
2192     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2193     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2194     Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2195     {\bbl@csarg\let{KVP@##1}\@nnil}%
2196   \global\let\bbl@release@transforms\@empty
2197   \global\let\bbl@release@casing\@empty
2198   \let\bbl@calendars\@empty
2199   \global\let\bbl@inidata\@empty
2200   \global\let\bbl@extend@ini@gobble
2201   \global\let\bbl@included@inis\@empty
2202   \gdef\bbl@key@list{;}%
2203   \bbl@ifunset{bbl@passto@#2}%
2204     {\def\bbl@tempa{#1}}%
2205     {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}}%
2206   \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2207     \in@{/}{##1}% With /, (re)sets a value in the ini
2208     \ifin@
2209       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2210       \bbl@renewinikey##1\@{##2}%
2211     \else
2212       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2213         \bbl@error{unknown-provide-key}{##1}{}{}%
2214       \fi
2215       \bbl@csarg\def{KVP@##1}{##2}%
2216     \fi}%
\end{package}
```

```

2217 \chardef\bbbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2218 \bbbl@ifunset{date#2}\z@{\bbbl@ifunset{bbbl@llevel@#2}\@ne\tw}%
2219 % == init ==
2220 \ifx\bbbl@screset\@undefined
2221 \bbbl@ldfinit
2222 \fi
2223 % ==
2224 \ifx\bbbl@KVP@@import\@nnil\else \ifx\bbbl@KVP@import\@nnil
2225 \def\bbbl@KVP@import{\@empty}%
2226 \fi\fi
2227 % == date (as option) ==
2228 % \ifx\bbbl@KVP@date\@nnil\else
2229 % \fi
2230 % ==
2231 \let\bbbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2232 \ifcase\bbbl@howloaded
2233 \let\bbbl@lbfkflag\@empty % new
2234 \else
2235 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2236 \let\bbbl@lbfkflag\@empty
2237 \fi
2238 \ifx\bbbl@KVP@import\@nnil\else
2239 \let\bbbl@lbfkflag\@empty
2240 \fi
2241 \fi
2242 % == import, captions ==
2243 \ifx\bbbl@KVP@import\@nnil\else
2244 \bbbl@exp{\bbbl@ifblank{\bbbl@KVP@import}}%
2245 {\ifx\bbbl@initload\relax
2246 \begingroup
2247 \def\BabelBeforeIni##1##2{\gdef\bbbl@KVP@import{##1}\endinput}%
2248 \bbbl@input@texini{##2}%
2249 \endgroup
2250 \else
2251 \xdef\bbbl@KVP@import{\bbbl@initload}%
2252 \fi}%
2253 {}%
2254 \let\bbbl@KVP@date\@empty
2255 \fi
2256 \let\bbbl@KVP@captions@@\bbbl@KVP@captions
2257 \ifx\bbbl@KVP@captions\@nnil
2258 \let\bbbl@KVP@captions\bbbl@KVP@import
2259 \fi
2260 % ==
2261 \ifx\bbbl@KVP@transforms\@nnil\else
2262 \bbbl@replace\bbbl@KVP@transforms{ },}%
2263 \fi
2264 % == Load ini ==
2265 \ifcase\bbbl@howloaded
2266 \bbbl@provide@new{##2}%
2267 \else
2268 \bbbl@ifblank{##1}%
2269 {}% With \bbbl@load@basic below
2270 {\bbbl@provide@renew{##2}}%
2271 \fi
2272 % == include == TODO
2273 % \ifx\bbbl@included@inis\@empty\else
2274 % \bbbl@replace\bbbl@included@inis{ },}%
2275 % \bbbl@foreach\bbbl@included@inis{%
2276 % \openin\bbbl@readstream=babel-##1.ini
2277 % \bbbl@extend@ini{##2}}%
2278 % \closein\bbbl@readstream
2279 % \fi

```



```

2280 % Post tasks
2281 % -----
2282 % == subsequent calls after the first provide for a locale ==
2283 \ifx\bbbl@inidata\@empty\else
2284   \bbbl@extend@ini{#2}%
2285 \fi
2286 % == ensure captions ==
2287 \ifx\bbbl@KVP@captions\@nnil\else
2288   \bbbl@ifunset{bbbl@extracaps@#2}%
2289     {\bbbl@exp{\\\babelensure[exclude=\\today]{#2}}}%
2290     {\bbbl@exp{\\\babelensure[exclude=\\today,
2291       include=\[bbbl@extracaps@#2]]{#2}}}%
2292   \bbbl@ifunset{bbbl@ensure@language}%
2293     {\bbbl@exp{%
2294       \\DeclareRobustCommand<bbbl@ensure@language>[1]{%
2295         \\foreignlanguage{language}%
2296         {###1}}}%
2297     }%
2298   \bbbl@exp{%
2299     \\bbbl@tglobal<bbbl@ensure@language>%
2300     \\bbbl@tglobal<bbbl@ensure@language\space>%
2301 \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.

```

2302 \bbbl@load@basic{#2}%
2303 % == script, language ==
2304 % Override the values from ini or defines them
2305 \ifx\bbbl@KVP@script\@nnil\else
2306   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2307 \fi
2308 \ifx\bbbl@KVP@language\@nnil\else
2309   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2310 \fi
2311 \ifcase\bbbl@engine\or
2312   \bbbl@ifunset{bbbl@chrng@language}{}%
2313   {\directlua{
2314     Babel.set_chranges_b('\bbbl@cl{sbc}', '\bbbl@cl{chrng}') }}%
2315 \fi
2316 % == Line breaking: intraspace, intrapenalty ==
2317 % For CJK, East Asian, Southeast Asian, if interspace in ini
2318 \ifx\bbbl@KVP@intraspace\@nnil\else % We can override the ini or set
2319   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2320 \fi
2321 \bbbl@provide@intraspace
2322 % == Line breaking: justification ==
2323 \ifx\bbbl@KVP@justification\@nnil\else
2324   \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2325 \fi
2326 \ifx\bbbl@KVP@linebreaking\@nnil\else
2327   \bbbl@xin@{\, \bbbl@KVP@linebreaking,}%
2328   {,elongated,kashida,cjk,padding,unhyphenated,}%
2329 \ifin@
2330   \bbbl@csarg\xdef
2331     {\lnbrk@language}{\expandafter\@car\bbbl@KVP@linebreaking\@nil}%
2332 \fi
2333 \fi
2334 \bbbl@xin@{/e}{\bbbl@cl{\lnbrk}}%
2335 \ifin@ \else \bbbl@xin@{/k}{\bbbl@cl{\lnbrk}} \fi
2336 \ifin@ \bbbl@arabicjust \fi
2337 % WIP
2338 \bbbl@xin@{/p}{\bbbl@cl{\lnbrk}}%

```

```

2339 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2340 % == Line breaking: hyphenate.other.(locale|script) ==
2341 \ifx\bbl@lbfkflag\empty
2342   \bbl@ifunset{bbl@hyotl@language}{%
2343     {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
2344       \bbl@startcommands*{language}{%
2345         \bbl@csarg\bbl@foreach{hyotl@language}{%
2346           \ifcase\bbl@engine
2347             \ifnum##1<257
2348               \SetHyphenMap{\BabelLower{##1}{##1}}%
2349             \fi
2350           \else
2351             \SetHyphenMap{\BabelLower{##1}{##1}}%
2352           \fi}%
2353         \bbl@endcommands}%
2354     \bbl@ifunset{bbl@hyots@language}{%
2355       {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2356         \bbl@csarg\bbl@foreach{hyots@language}{%
2357           \ifcase\bbl@engine
2358             \ifnum##1<257
2359               \global\lccode##1=##1\relax
2360             \fi
2361           \else
2362             \global\lccode##1=##1\relax
2363           \fi}}%
2364     \fi
2365 % == Counters: maparabic ==
2366 % Native digits, if provided in ini (TeX level, xe and lua)
2367 \ifcase\bbl@engine\else
2368   \bbl@ifunset{bbl@dgnat@language}{%
2369     {\expandafter\ifx\csname bbl@dgnat@language\endcsname\empty\else
2370       \expandafter\expandafter\expandafter
2371       \bbl@setdigits\csname bbl@dgnat@language\endcsname
2372       \ifx\bbl@KVP@maparabic\@nnil\else
2373         \ifx\bbl@latinarabic\@undefined
2374           \expandafter\let\expandafter\@arabic
2375           \csname bbl@counter@language\endcsname
2376         \else % i.e., if layout=counters, which redefines \@arabic
2377           \expandafter\let\expandafter\bbl@latinarabic
2378           \csname bbl@counter@language\endcsname
2379         \fi
2380       \fi
2381     \fi}%
2382 \fi
2383 % == Counters: mapdigits ==
2384 % > luababel.def
2385 % == Counters: alph, Alph ==
2386 \ifx\bbl@KVP@alph\@nnil\else
2387   \bbl@exp{%
2388     \\bbl@add<bbl@preextras@language>{%
2389       \\babel@save\\@alph
2390       \let\\@alph<bbl@cntr@bbl@KVP@alph @language>}}%
2391 \fi
2392 \ifx\bbl@KVP@Alph\@nnil\else
2393   \bbl@exp{%
2394     \\bbl@add<bbl@preextras@language>{%
2395       \\babel@save\\@Alph
2396       \let\\@Alph<bbl@cntr@bbl@KVP@Alph @language>}}%
2397 \fi
2398 % == Casing ==
2399 \bbl@release@casing
2400 \ifx\bbl@KVP@casing\@nnil\else
2401   \bbl@csarg\xdef{casing@language}%

```

```

2402     {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2403 \fi
2404 % == Calendars ==
2405 \ifx\bbl@KVP@calendar\@nnil
2406   \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2407 \fi
2408 \def\bbl@tempe##1 ##2\@{% % Get first calendar
2409   \def\bbl@tempa{##1}%
2410   \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\\ \@}%
2411 \def\bbl@tempe##1.##2.##3\@{%
2412   \def\bbl@tempc{##1}%
2413   \def\bbl@tempb{##2}}%
2414 \expandafter\bbl@tempe\bbl@tempa.\@
2415 \bbl@csarg\edef{calpr@\languagename}{%
2416   \ifx\bbl@tempc@empty\else
2417     calendar=\bbl@tempc
2418   \fi
2419   \ifx\bbl@tempb@empty\else
2420     ,variant=\bbl@tempb
2421   \fi}%
2422 % == engine specific extensions ==
2423 % Defined in XXXbabel.def
2424 \bbl@provide@extra{#2}%
2425 % == require.babel in ini ==
2426 % To load or reload the babel-*.tex, if require.babel in ini
2427 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2428   \bbl@ifunset{\bbl@rqtex@\languagename}{}%
2429   {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2430     \let\BabelBeforeIni@gobbletwo
2431     \chardef\atcatcode=\catcode` \@
2432     \catcode`\@=11\relax
2433     \def\CurrentOption{#2}%
2434     \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2435     \catcode`\@=\atcatcode
2436     \let\atcatcode\relax
2437     \global\bbl@csarg\let{rqtex@\languagename}\relax
2438   \fi}%
2439 \bbl@foreach\bbl@calendars{%
2440   \bbl@ifunset{\bbl@ca##1}{%
2441     \chardef\atcatcode=\catcode` \@
2442     \catcode`\@=11\relax
2443     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2444     \catcode`\@=\atcatcode
2445     \let\atcatcode\relax}%
2446   {}}%
2447 \fi
2448 % == frenchspacing ==
2449 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2450 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2451 \ifin@
2452   \bbl@extras@wrap{\bbl@pre@fs}%
2453   {\bbl@pre@fs}%
2454   {\bbl@post@fs}%
2455 \fi
2456 % == transforms ==
2457 % > luababel.def
2458 \def\CurrentOption{#2}%
2459 \@nameuse{bbl@icsave@#2}%
2460 % == main ==
2461 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2462   \let\languagename\bbl@savelangname
2463   \chardef\localeid\bbl@savelocaleid\relax
2464 \fi

```

```

2465 % == hyphenrules (apply if current) ==
2466 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2467   \ifnum\bbbl@savetoday=\today
2468     \language\@nameuse{l@language}%
2469   \fi
2470 \fi}

  Depending on whether or not the language exists (based on \date{language}), we define two
  macros. Remember \bbbl@startcommands opens a group.

2471 \def\bbbl@provide@new#1{%
2472   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2473   \@namedef{extras#1}{}%
2474   \@namedef{noextras#1}{}%
2475   \bbbl@startcommands*{#1}{captions}%
2476   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2477     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2478       \ifx##1\@nnil\else
2479         \bbbl@exp{%
2480           \SetString\##1{%
2481             \bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}%
2482             \expandafter\bbbl@tempb
2483           \fi}%
2484       \expandafter\bbbl@tempb\bbbl@captionslist\@nnil
2485     \else
2486       \ifx\bbbl@initload\relax
2487         \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2488       \else
2489         \bbbl@read@ini{\bbbl@initload}2% % Same
2490       \fi
2491     \fi
2492   \StartBabelCommands*{#1}{date}%
2493   \ifx\bbbl@KVP@date\@nnil
2494     \bbbl@exp{%
2495       \SetString\today{\bbbl@nocaption{today}{#1today}}}%
2496   \else
2497     \bbbl@savetoday
2498     \bbbl@savetoday
2499   \fi
2500 \bbbl@endcommands
2501 \bbbl@load@basic{#1}%
2502 % == hyphenmins == (only if new)
2503 \bbbl@exp{%
2504   \gdef\<#1hyphenmins>{%
2505     {\bbbl@ifunset{\bbbl@lftm#1}{2}{\bbbl@cs{lftm#1}}}%
2506     {\bbbl@ifunset{\bbbl@rgtm#1}{3}{\bbbl@cs{rgtm#1}}}%
2507   % == hyphenrules (also in renew) ==
2508   \bbbl@provide@hyphens{#1}%
2509   \ifx\bbbl@KVP@main\@nnil\else
2510     \expandafter\main@language\expandafter{#1}%
2511   \fi}
2512 %
2513 \def\bbbl@provide@renew#1{%
2514   \ifx\bbbl@KVP@captions\@nnil\else
2515     \StartBabelCommands*{#1}{captions}%
2516     \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2517     \EndBabelCommands
2518   \fi
2519   \ifx\bbbl@KVP@date\@nnil\else
2520     \StartBabelCommands*{#1}{date}%
2521     \bbbl@savetoday
2522     \bbbl@savetoday
2523     \EndBabelCommands
2524   \fi

```

```

2525 % == hyphenrules (also in new) ==
2526 \ifx\bbbl@lbfkflag\empty
2527   \bbbl@provide@hyphens{#1}%
2528 \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.

```

2529 \def\bbbl@load@basic#1{%
2530   \ifcase\bbbl@howloaded\or\or
2531     \ifcase\csname bbl@llevel@\language\endcsname
2532       \bbbl@csarg\let\lname@\language\relax
2533     \fi
2534   \fi
2535   \bbbl@ifunset{bbbl@lname@#1}%
2536   {\def\BabelBeforeIni##1##2{%
2537     \begingroup
2538       \let\bbbl@ini@captions@aux\@gobbletwo
2539       \def\bbbl@inidate #####1.####2.####3.####4\relax #####5####6{%
2540         \bbbl@read@ini{##1}l%
2541         \ifx\bbbl@initoload\relax\endinput\fi
2542       \endgroup}%
2543     \begingroup      % boxed, to avoid extra spaces:
2544     \ifx\bbbl@initoload\relax
2545       \bbbl@input@texini{#1}%
2546     \else
2547       \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
2548     \fi
2549   \endgroup}%
2550   {}}

```

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.

```

2551 \def\bbbl@provide@hyphens#1{%
2552   \@tempcnta\m@ne % a flag
2553   \ifx\bbbl@KVP@hyphenrules\@nnil\else
2554     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
2555     \bbbl@foreach\bbbl@KVP@hyphenrules{%
2556       \ifnum\@tempcnta=\m@ne % if not yet found
2557         \bbbl@ifsamestring{##1}{+}%
2558         {\bbbl@carg\addlanguage{l@##1}}%
2559         {}%
2560         \bbbl@ifunset{l@##1}% After a possible +
2561         {}%
2562         {\@tempcnta\@nameuse{l@##1}}%
2563       \fi}%
2564   \ifnum\@tempcnta=\m@ne
2565     \bbbl@warning{%
2566       Requested 'hyphenrules' for '\language' not found:\\%
2567       \bbbl@KVP@hyphenrules.\\%
2568       Using the default value. Reported}%
2569   \fi
2570 \fi
2571 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2572   \ifx\bbbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2573     \bbbl@ifunset{bbbl@hyphr@#1}{% use value in ini, if exists
2574       {\bbbl@exp{\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
2575       {}%
2576       {\bbbl@ifunset{l@bbbl@cl{hyphr}}}%
2577       {}% if hyphenrules found:
2578       {\@tempcnta\@nameuse{l@bbbl@cl{hyphr}}}}}%
2579   \fi
2580 \fi
2581 \bbbl@ifunset{l@#1}%

```

```

2582     {\ifnum\@tempcnta=\m@ne
2583       \bbl@carg\adddialect{l@#1}\language
2584     \else
2585       \bbl@carg\adddialect{l@#1}\@tempcnta
2586     \fi}%
2587 {\ifnum\@tempcnta=\m@ne\else
2588   \global\bbl@carg\chardef{l@#1}\@tempcnta
2589   \fi}}

```

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

```

2590 \def\bbl@input@texini#1{%
2591   \bbl@bsphack
2592   \bbl@exp{%
2593     \catcode`\\=14 \catcode`\\=0
2594     \catcode`\\={1 \catcode`\\}=2
2595     \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2596     \catcode`\\=\the\catcode`\relax
2597     \catcode`\\=\the\catcode`\relax
2598     \catcode`\\={\the\catcode`\relax
2599     \catcode`\\=\the\catcode`\relax}%
2600   \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.

```

2601 \def\bbl@iniline#1\bbl@iniline{%
2602   \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2603 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2604 \def\bbl@iniskip#1\@@{%      if starts with ;
2605 \def\bbl@inistore#1=#2\@@{%  full (default)
2606   \bbl@trim@def\bbl@tempa{#1}%
2607   \bbl@trim\toks@{#2}%
2608   \bbl@ifsamestring{\bbl@tempa}{\include}%
2609   {\bbl@read@subini{\the\toks@}}%
2610   {\bbl@xin@{\bbl@section/\bbl@tempa}{\bbl@key@list}%
2611    \ifin@ \else
2612      \bbl@xin@{,identification/include.}%
2613      {,\bbl@section/\bbl@tempa}%
2614      \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2615      \bbl@exp{%
2616        \\g@addto@macro\\bbl@inidata{%
2617          \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2618      \fi}}
2619 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2620   \bbl@trim@def\bbl@tempa{#1}%
2621   \bbl@trim\toks@{#2}%
2622   \bbl@xin@{.identification.}{.\bbl@section.}%
2623   \ifin@
2624     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2625       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2626   \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.

```

2627 \def\bbl@loop@ini#1{%
2628   \loop
2629     \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2630     \endlinechar\m@ne
2631     \read#1 to \bbl@line
2632     \endlinechar\^^M
2633     \ifx\bbl@line\empty\else
2634       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2635     \fi
2636   \repeat}
2637 \def\bbl@read@subini#1{%
2638   \ifx\bbl@readsubstream\undefined
2639     \csname newread\endcsname\bbl@readsubstream
2640   \fi
2641   \openin\bbl@readsubstream=babel-#1.ini
2642   \ifeof\bbl@readsubstream
2643     \bbl@error{no-ini-file}{#1}{}{}%
2644   \else
2645     {\bbl@loop@ini\bbl@readsubstream}%
2646   \fi
2647   \closein\bbl@readsubstream}
2648 \ifx\bbl@readstream\undefined
2649   \csname newread\endcsname\bbl@readstream
2650 \fi
2651 \def\bbl@read@ini#1#2{%
2652   \global\let\bbl@extend@ini@gobble
2653   \openin\bbl@readstream=babel-#1.ini
2654   \ifeof\bbl@readstream
2655     \bbl@error{no-ini-file}{#1}{}{}%
2656   \else
2657     % == Store ini data in \bbl@inidata ==
2658     \catcode\ [=12 \catcode\ ]=12 \catcode\ ==12 \catcode\ &=12
2659     \catcode\ ;=12 \catcode\ |=12 \catcode\ %=14 \catcode\ -=12
2660     \ifnum#2=\m@ne % Just for the info
2661       \edef\language\tag\bbl@metalang}%
2662     \fi
2663     \bbl@info{Importing
2664               \ifcase#2font and identification \or basic \fi
2665               data for \language\}%
2666               from babel-#1.ini. Reported}%
2667   \ifnum#2<\@ne
2668     \global\let\bbl@inidata\empty
2669     \let\bbl@inistore\bbl@inistore@min % Remember it's local
2670   \fi
2671   \def\bbl@section{identification}%
2672   \bbl@exp{%
2673     \\\bbl@inistore tag.ini=#1\\@@
2674     \\\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\@@}%
2675   \bbl@loop@ini\bbl@readstream
2676   % == Process stored data ==
2677   \ifnum#2=\m@ne
2678     \def\bbl@tempa##1 ##2\@@{##1}% Get first name
2679     \def\bbl@elt##1##2##3{%
2680       \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2681       {\xdef\language{\bbl@tempa##3 \@@}%
2682        \bbl@id@assign
2683        \def\bbl@elt####1####2####3{}}}%
2684     {}}%

```

```

2685 \bbl@inidata
2686 \fi
2687 \bbl@csarg\xdef{lini@\languagename}{#1}%
2688 \bbl@read@ini@aux
2689 % == 'Export' data ==
2690 \bbl@ini@exports{#2}%
2691 \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2692 \global\let\bbl@inidata\@empty
2693 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
2694 \bbl@tglobal\bbl@ini@loaded
2695 \fi
2696 \closein\bbl@readstream}
2697 \def\bbl@read@ini@aux{%
2698 \let\bbl@savestrings\@empty
2699 \let\bbl@savetoday\@empty
2700 \let\bbl@savestate\@empty
2701 \def\bbl@elt##1##2##3{%
2702 \def\bbl@section{##1}%
2703 \in@{=date.}{=##1}% Find a better place
2704 \ifin@
2705 \bbl@ifunset{bbl@inikv@##1}%
2706 {\bbl@ini@calendar{##1}}%
2707 {}%
2708 \fi
2709 \bbl@ifunset{bbl@inikv@##1}{}%
2710 {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2711 \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.

```

2712 \def\bbl@extend@ini@aux#1{%
2713 \bbl@startcommands*{#1}{captions}%
2714 % Activate captions/... and modify exports
2715 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2716 \setlocalecaption{#1}{##1}{##2}}%
2717 \def\bbl@inikv@captions##1##2{%
2718 \bbl@ini@captions@aux{##1}{##2}}%
2719 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2720 \def\bbl@exportkey##1##2##3{%
2721 \bbl@ifunset{bbl@kv@##2}{}%
2722 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2723 \bbl@exp{\global\let<bbl@##1@\languagename>\<bbl@kv@##2>}}%
2724 \fi}}%
2725 % As with \bbl@read@ini, but with some changes
2726 \bbl@read@ini@aux
2727 \bbl@ini@exports\tw@
2728 % Update inidata@lang by pretending the ini is read.
2729 \def\bbl@elt##1##2##3{%
2730 \def\bbl@section{##1}%
2731 \bbl@iniline##2=##3\bbl@iniline}%
2732 \csname bbl@inidata@#1\endcsname
2733 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2734 \StartBabelCommands*{#1}{date}% And from the import stuff
2735 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2736 \bbl@savetoday
2737 \bbl@savestate
2738 \bbl@endcommands}

```

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

```

2739 \def\bbl@ini@calendar#1{%
2740 \lowercase{\def\bbl@tempa{=#1=}}%
2741 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2742 \bbl@replace\bbl@tempa{=date.}{}%
2743 \in@{.licr=}{#1=}%

```



```

2744 \ifin@
2745 \ifcase\bbl@engine
2746 \bbl@replace\bbl@tempa{.licr={}}}%
2747 \else
2748 \let\bbl@tempa\relax
2749 \fi
2750 \fi
2751 \ifx\bbl@tempa\relax\else
2752 \bbl@replace\bbl@tempa{=}{}}}%
2753 \ifx\bbl@tempa\@empty\else
2754 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2755 \fi
2756 \bbl@exp{%
2757 \def\bbl@inikv@#1>###1###2{%
2758 \\\bbl@inidate###1...\relax{###2}{\bbl@tempa}}}%
2759 \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).

```

2760 \def\bbl@renewinikey#1/#2\@#3{%
2761 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2762 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2763 \bbl@trim\toks@{#3}% value
2764 \bbl@exp{%
2765 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2766 \\g@addto@macro\\bbl@inidata{%
2767 \\\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.

```

2768 \def\bbl@exportkey#1#2#3{%
2769 \bbl@ifunset{\bbl@kv@#2}%
2770 {\bbl@csarg\gdef{#1@\language name}{#3}}%
2771 {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2772 \bbl@csarg\gdef{#1@\language name}{#3}%
2773 \else
2774 \bbl@exp{\global\let<\bbl@#1@\language name>\<\bbl@kv@#2>}%
2775 \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.

```

2776 \def\bbl@iniwarning#1{%
2777 \bbl@ifunset{\bbl@kv@identification.warning#1}{}}%
2778 {\bbl@warning{%
2779 From babel-\bbl@cs{lini@\language name}.ini:\\%
2780 \bbl@cs{kv@identification.warning#1}\\%
2781 Reported }}}
2782 %
2783 \let\bbl@release@transforms\@empty
2784 \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).

```

2785 \def\bbl@ini@exports#l{%
2786   % Identification always exported
2787   \bbl@iniwarning{}%
2788   \ifcase\bbl@engine
2789     \bbl@iniwarning{.pdflatex}%
2790   \or
2791     \bbl@iniwarning{.lualatex}%
2792   \or
2793     \bbl@iniwarning{.xelatex}%
2794   \fi%
2795   \bbl@exportkey{lllevel}{identification.load.level}{}%
2796   \bbl@exportkey{elname}{identification.name.english}{}%
2797   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2798     {\csname bbl@elname@languagename\endcsname}}%
2799   \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2800   % Somewhat hackish. TODO:
2801   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2802   \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2803   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2804   \bbl@exportkey{esname}{identification.script.name}{}%
2805   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2806     {\csname bbl@esname@languagename\endcsname}}%
2807   \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2808   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2809   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2810   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2811   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2812   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2813   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2814   % Also maps bcp47 -> languagename
2815   \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{\languagename}%
2816   \ifcase\bbl@engine\or
2817     \directlua{%
2818       Babel.locale_props[\the\bbl@cs{id@languagename}].script
2819       = '\bbl@cl{sbcpl}'}%
2820   \fi
2821   % Conditional
2822   \ifnum#l>\z@      % -1 or 0 = only info, 1 = basic, 2 = (re)new
2823     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2824     \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
2825     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2826     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2827     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2828     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2829     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2830     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2831     \bbl@exportkey{intsp}{typography.intraspace}{}%
2832     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2833     \bbl@exportkey{chrng}{characters.ranges}{}%
2834     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2835     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2836     \ifnum#l=\tw@      % only (re)new
2837       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2838       \bbl@tglobal\bbl@savetoday
2839       \bbl@tglobal\bbl@savestate
2840       \bbl@savestrings
2841     \fi
2842   \fi}

```

4.20. Processing keys in ini

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

```
2843 \def\bbl@inikv#1#2{%      key=value
2844   \toks@{#2}%              This hides #'s from ini values
2845   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
2846 \let\bbl@inikv@identification\bbl@inikv
2847 \let\bbl@inikv@date\bbl@inikv
2848 \let\bbl@inikv@typography\bbl@inikv
2849 \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.

```
2850 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\empty x-\fi}
2851 \def\bbl@inikv@characters#1#2{%
2852   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2853   {\bbl@exp{%
2854     \\g@addto@macro\\bbl@release@casing{%
2855       \\bbl@casemapping}{\language}{\unexpanded{#2}}}%
2856   {\in@{casing.}{#1}% e.g., casing.Uv = uV
2857   \ifin@
2858     \lowercase{\def\bbl@tempb{#1}%
2859     \bbl@replace\bbl@tempb{casing.}{}%
2860     \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2861       \\bbl@casemapping
2862       {\\bbl@maybextx\bbl@tempb}{\language}{\unexpanded{#2}}}%
2863     \else
2864     \bbl@inikv{#1}{#2}%
2865     \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’.

```
2866 \def\bbl@inikv@counters#1#2{%
2867   \bbl@ifsamestring{#1}{digits}%
2868   {\bbl@error{digits-is-reserved}{}}}%
2869   {%
2870   \def\bbl@tempc{#1}%
2871   \bbl@trim@def{\bbl@tempb*}{#2}%
2872   \in@{.1$}{#1$}%
2873   \ifin@
2874     \bbl@replace\bbl@tempc{.1}{}%
2875     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
2876     \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2877   \fi
2878   \in@{.F.}{#1}%
2879   \ifin@\else\in@{.S.}{#1}\fi
2880   \ifin@
2881     \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
2882   \else
2883     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2884     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2885     \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
2886     \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.

```
2887 \ifcase\bbl@engine
2888   \bbl@csarg\def{inikv@captions.licr}#1#2{%
2889     \bbl@ini@captions@aux{#1}{#2}}
```

```

2890 \else
2891   \def\bbl@inikv@captions#1#2{%
2892     \bbl@ini@captions@aux#{#1}{#2}}
2893 \fi

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

2894 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2895   \bbl@replace\bbl@tempa{.template}{}%
2896   \def\bbl@toreplace{#1}{}%
2897   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2898   \bbl@replace\bbl@toreplace{[{}]{\csname}%
2899   \bbl@replace\bbl@toreplace{[{}]{\csname the}%
2900   \bbl@replace\bbl@toreplace{[{}]{name\endcsname{}}}%
2901   \bbl@replace\bbl@toreplace{[{}]{\endcsname{}}}%
2902   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2903   \ifin@
2904     \@nameuse{\bbl@patch\bbl@tempa}%
2905     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2906   \fi
2907   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2908   \ifin@
2909     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2910     \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2911       \\bbl@ifunset{\bbl@bbl@tempa fmt@\\language}%
2912       {\[fnum@\bbl@tempa]}%
2913       {\@nameuse{\bbl@bbl@tempa fmt@\\language}}}%
2914   \fi}
2915 \def\bbl@ini@captions@aux#1#2{%
2916   \bbl@trim@def\bbl@tempa{#1}%
2917   \bbl@xin@{.template}{\bbl@tempa}%
2918   \ifin@
2919     \bbl@ini@captions@template{#2}\language
2920   \else
2921     \bbl@ifblank{#2}%
2922     {\bbl@exp{%
2923       \toks@{\@nameuse{\bbl@nocaption{\bbl@tempa}\language\bbl@tempa name}}}%
2924     {\bbl@trim\toks@{#2}}}%
2925     \bbl@exp{%
2926       \\bbl@add\\bbl@savestrings{%
2927         \\SetString\<\bbl@tempa name>{\the\toks@}}%
2928       \toks@\expandafter{\bbl@captionslist}%
2929       \bbl@exp{\@nameuse{\<\bbl@tempa name>}\the\toks@}}%
2930     \ifin@else
2931       \bbl@exp{%
2932         \\bbl@add\<\bbl@extracaps@language>{\<\bbl@tempa name>}%
2933         \\bbl@togetglobal\<\bbl@extracaps@language>}%
2934     \fi
2935   \fi}

```

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

```

2936 \def\bbl@list@the{%
2937   part,chapter,section,subsection,subsubsection,paragraph,%
2938   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2939   table,page,footnote,mpfootnote,mpfn}
2940 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2941   \bbl@ifunset{\bbl@map@#1@language}%
2942   {\@nameuse{#1}}%
2943   {\@nameuse{\bbl@map@#1@language}}}
2944 \def\bbl@inikv@labels#1#2{%
2945   \in@{.map}{#1}%
2946   \ifin@
2947     \ifx\bbl@KVP@labels\@nnil\else
2948       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
2949     \ifin@

```

```

2950 \def\bbl@tempc{#1}%
2951 \bbl@replace\bbl@tempc{.map}{}%
2952 \in@{, #2, }{, arabic, roman, Roman, alph, Alph, fnsymbol,}%
2953 \bbl@exp{%
2954 \gdef\<bbl@map@bbl@tempc @\language\name>%
2955 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
2956 \bbl@foreach\bbl@list@the{%
2957 \bbl@ifunset{the##1}{}%
2958 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
2959 \bbl@exp{%
2960 \\bbl@sreplace\<the##1>%
2961 {\<\bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}%
2962 \\bbl@sreplace\<the##1>%
2963 {\<\empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2964 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2965 \toks@ \expandafter\expandafter\expandafter{%
2966 \csname the##1\endcsname}%
2967 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
2968 \fi}}%
2969 \fi
2970 \fi
2971 %
2972 \else
2973 %
2974 % The following code is still under study. You can test it and make
2975 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
2976 % language dependent.
2977 \in@{enumerate.}{#1}%
2978 \ifin@
2979 \def\bbl@tempa{#1}%
2980 \bbl@replace\bbl@tempa{enumerate.}{}%
2981 \def\bbl@toreplace{#2}%
2982 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2983 \bbl@replace\bbl@toreplace{[]}{\csname the}%
2984 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
2985 \toks@ \expandafter{\bbl@toreplace}%
2986 % TODO. Execute only once:
2987 \bbl@exp{%
2988 \\bbl@add\<extras\language\name>{%
2989 \\babel@save\<labelenum\romannumeral\bbl@tempa>%
2990 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
2991 \\bbl@tglobal\<extras\language\name>}%
2992 \fi
2993 \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.

```

2994 \def\bbl@chapttype{chapter}
2995 \ifx\@makechapterhead\@undefined
2996 \let\bbl@patchchapter\relax
2997 \else\ifx\thechapter\@undefined
2998 \let\bbl@patchchapter\relax
2999 \else\ifx\ps@headings\@undefined
3000 \let\bbl@patchchapter\relax
3001 \else
3002 \def\bbl@patchchapter{%
3003 \global\let\bbl@patchchapter\relax
3004 \gdef\bbl@chfmt{%
3005 \bbl@ifunset\bbl@\bbl@chapttype fmt@\language\name}%
3006 {\@chapapp\space\thechapter}%
3007 {\@nameuse\bbl@\bbl@chapttype fmt@\language\name}}%

```

```

3008 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3009 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3010 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3011 \bbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3012 \bbl@toglobal\appendix
3013 \bbl@toglobal\ps@headings
3014 \bbl@toglobal\chaptermark
3015 \bbl@toglobal\makechapterhead}
3016 \let\bbl@patchappendix\bbl@patchchapter
3017 \fi\fi\fi
3018 \ifx\@part\@undefined
3019 \let\bbl@patchpart\relax
3020 \else
3021 \def\bbl@patchpart{%
3022 \global\let\bbl@patchpart\relax
3023 \gdef\bbl@partformat{%
3024 \bbl@ifunset{bbl@partfmt@\language}%
3025 {\partname\nobreakspace\thepart}%
3026 {\@nameuse{bbl@partfmt@\language}}}%
3027 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3028 \bbl@toglobal\@part}
3029 \fi

```

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

```

3030 \let\bbl@calendar\@empty
3031 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3032 \def\bbl@localedate#1#2#3#4{%
3033 \begingroup
3034 \edef\bbl@they{#2}%
3035 \edef\bbl@them{#3}%
3036 \edef\bbl@thed{#4}%
3037 \edef\bbl@tempe{%
3038 \bbl@ifunset{bbl@calpr@\language}{\bbl@cl@calpr}},%
3039 #1}%
3040 \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3041 \bbl@replace\bbl@tempe{ }{}%
3042 \bbl@replace\bbl@tempe{convert}{convert=}%
3043 \let\bbl@ld@calendar\@empty
3044 \let\bbl@ld@variant\@empty
3045 \let\bbl@ld@convert\relax
3046 \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld##1}{##2}}%
3047 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3048 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3049 \ifx\bbl@ld@calendar\@empty\else
3050 \ifx\bbl@ld@convert\relax\else
3051 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3052 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3053 \fi
3054 \fi
3055 \@nameuse{bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3056 \edef\bbl@calendar{% Used in \month..., too
3057 \bbl@ld@calendar
3058 \ifx\bbl@ld@variant\@empty\else
3059 .\bbl@ld@variant
3060 \fi}%
3061 \bbl@cased
3062 {\@nameuse{bbl@date@\language @\bbl@calendar}%
3063 \bbl@they\bbl@them\bbl@thed}%
3064 \endgroup}
3065 \def\bbl@printdate#1{%
3066 \@ifnextchar[\bbl@printdate@i{#1}]{\bbl@printdate@i{#1}[]}}
3067 \def\bbl@printdate@i#1[#2]#3#4#5{%

```

```

3068 \bbl@usedategroupttrue
3069 \@nameuse{bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}
3070 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3071 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3072 \bbl@trim@def\bbl@tempa{#1.#2}%
3073 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3074 {\bbl@trim@def\bbl@tempa{#3}%
3075 \bbl@trim\toks@{#5}%
3076 \@temptokena\expandafter{\bbl@savedate}%
3077 \bbl@exp{% Reverse order - in ini last wins
3078 \def\\bbl@savedate{%
3079 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3080 \the\@temptokena}}}%
3081 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3082 {\lowercase{\def\bbl@tempb{#6}}}%
3083 \bbl@trim@def\bbl@toreplace{#5}%
3084 \bbl@TG@@date
3085 \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3086 \ifx\bbl@savetoday\@empty
3087 \bbl@exp{% TODO. Move to a better place.
3088 \\AfterBabelCommands{%
3089 \gdef\<\language name date>{\protect\<\language name date >}%
3090 \gdef\<\language name date >{\bbl@printdate{\language name}}}%
3091 \def\\bbl@savetoday{%
3092 \\SetString\\today{%
3093 \<\language name date>[convert]%
3094 {\the\year}{\the\month}{\the\day}}}%
3095 \fi}%
3096 {}}}}

```

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).

```

3097 \let\bbl@calendar\@empty
3098 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3099 \@nameuse{bbl@ca@#2}#1\@@}
3100 \newcommand\babelDateSpace{\nobreakspace}
3101 \newcommand\babelDateDot{.\@}
3102 \newcommand\babelDated[1]{\number#1}
3103 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3104 \newcommand\babelDateM[1]{\number#1}
3105 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3106 \newcommand\babelDateMMM[1]{%
3107 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3108 \newcommand\babelDatey[1]{\number#1}%
3109 \newcommand\babelDateyy[1]{%
3110 \ifnum#1<10 0\number#1 %
3111 \else\ifnum#1<100 \number#1 %
3112 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3113 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3114 \else
3115 \bbl@error{limit-two-digits}{\number#1}%
3116 \fi\fi\fi\fi}
3117 \newcommand\babelDateyyyy[1]{\number#1} % TODO - add leading 0
3118 \newcommand\babelDateU[1]{\number#1}%
3119 \def\bbl@replace@finish@iii#1{%
3120 \bbl@exp{\def\\#1###1###2###3{\the\toks@}}
3121 \def\bbl@TG@@date{%
3122 \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3123 \bbl@replace\bbl@toreplace{.}{\babelDateDot}}%
3124 \bbl@replace\bbl@toreplace{[d]}{\babelDated{###3}}%

```

```

3125 \bbl@replace\bbl@toreplace{[dd]}\BabelDatedd{####3}%
3126 \bbl@replace\bbl@toreplace{[M]}\BabelDateM{####2}%
3127 \bbl@replace\bbl@toreplace{[MM]}\BabelDateMM{####2}%
3128 \bbl@replace\bbl@toreplace{[MMMM]}\BabelDateMMMM{####2}%
3129 \bbl@replace\bbl@toreplace{[y]}\BabelDatey{####1}%
3130 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{####1}%
3131 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}%
3132 \bbl@replace\bbl@toreplace{[U]}\BabelDateU{####1}%
3133 \bbl@replace\bbl@toreplace{[y]}\bbl@datecctr{####1}%
3134 \bbl@replace\bbl@toreplace{[U]}\bbl@datecctr{####1}%
3135 \bbl@replace\bbl@toreplace{[m]}\bbl@datecctr{####2}%
3136 \bbl@replace\bbl@toreplace{[d]}\bbl@datecctr{####3}%
3137 \bbl@replace@finish@iii\bbl@toreplace}
3138 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3139 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

4.21. French spacing (again)

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

```

3140 \AddToHook{begindocument/before}{%
3141   \let\bbl@normalsf\normalsfcodes
3142   \let\normalsfcodes\relax}
3143 \AtBeginDocument{%
3144   \ifx\bbl@normalsf\empty
3145     \ifnum\scode\.\@m
3146       \let\normalsfcodes\frenchspacing
3147     \else
3148       \let\normalsfcodes\nonfrenchspacing
3149     \fi
3150   \else
3151     \let\normalsfcodes\bbl@normalsf
3152   \fi}

```

Transforms.

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

```

3153 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3154 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3155 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3156   #1[#2]{#3}{#4}{#5}}
3157 \begin{group}
3158   \catcode\%=12
3159   \catcode\&=14
3160   \gdef\bbl@transforms#1#2#3{%&
3161     \directlua{
3162       local str = [==[#2]==]
3163       str = str:gsub('%d+%.%d+$', '')
3164       token.set_macro('babeltempa', str)
3165     }&
3166     \def\babeltempc{}&
3167     \bbl@xin@{\babeltempa,}{\bbl@KVP@transforms,}&
3168     \ifin@\else
3169       \bbl@xin@{\babeltempa,}{\bbl@KVP@transforms,}&
3170     \fi
3171     \ifin@
3172       \bbl@foreach\bbl@KVP@transforms{%&
3173         \bbl@xin@{\babeltempa,}{,##1,}&
3174         \ifin@ & font:font:transform syntax
3175       \directlua{

```



```

3176         local t = {}
3177         for m in string.gmatch('##1'..' ':'', '(.-):') do
3178             table.insert(t, m)
3179         end
3180         table.remove(t)
3181         token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3182     }&%
3183     \fi}&%
3184     \in@{.0$}{#2$}&%
3185     \ifin@
3186     \directlua{&% (\attribute) syntax
3187         local str = string.match([[ \bbl@KVP@transforms]],
3188             '%([^(%-)]-)%([%)]-)\babeltempa')
3189         if str == nil then
3190             token.set_macro('babeltempb', '')
3191         else
3192             token.set_macro('babeltempb', ', attribute=' .. str)
3193         end
3194     }&%
3195     \toks@{#3}&%
3196     \bbl@exp{&%
3197         \\g@addto@macro\\ \bbl@release@transforms{&%
3198             \relax &% Closes previous \bbl@transforms@aux
3199             \\ \bbl@transforms@aux
3200             \\ #1{label=\babeltempa\babeltempb\babeltempc}&%
3201             {\language\the\toks@}}&%
3202     \else
3203         \g@addto@macro \bbl@release@transforms{, {#3}}&%
3204     \fi
3205     \fi}
3206 \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.

```

3207 \def\bbl@provide@lsys#1{%
3208     \bbl@ifunset{\bbl@lname@#1}%
3209     {\bbl@load@info{#1}}%
3210     {%
3211         \bbl@csarg\let{lsys@#1}\empty
3212         \bbl@ifunset{\bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{%
3213             \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3214                 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3215                 \bbl@ifunset{\bbl@lname@#1}{%
3216                     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3217                 \ifcase\bbl@engine\or\or
3218                 \bbl@ifunset{\bbl@prehc@#1}{%
3219                     {\bbl@exp{\\ \bbl@ifblank{\bbl@cs{prehc@#1}}}%
3220                     }%
3221                     {\ifx\bbl@xenoxyph\undefined
3222                         \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3223                         \ifx\AtBeginDocument\@notprerr
3224                             \expandafter\@secondoftwo % to execute right now
3225                         \fi
3226                         \AtBeginDocument{%
3227                             \bbl@patchfont{\bbl@xenoxyph}%
3228                             {\expandafter\select@language\expandafter{\language\the\toks@}}%
3229                         \fi}}%
3230         \fi
3231         \bbl@csarg\bbl@toglobal{lsys@#1}}

```

```

3232 \def\bbl@load@info#1{%
3233   \def\BabelBeforeIni##1##2{%
3234     \begingroup
3235       \bbl@read@ini{##1}0%
3236       \endinput           % babel- .tex may contain onlypreamble's
3237       \endgroup}%         boxed, to avoid extra spaces:
3238   {\bbl@input@texini{#1}}}
```

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

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

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before \@@ collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210.

Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3279 \newcommand\locaenumerat[2]{\bbl@cs{cnt@#1\@language}{#2}}
3280 \def\bbl@locaecnt#1#2{\locaenumerat{#2}{#1}}
3281 \newcommand\locaecounter[2]{%
3282   \expandafter\bbl@locaecnt
3283   \expandafter\@number\csname c@#2\endcsname}{#1}}
3284 \def\bbl@alphnumerat#1#2{%
3285   \expandafter\bbl@alphnumerat@i\@number#2 76543210\@{#1}}
3286 \def\bbl@alphnumerat@i#1#2#3#4#5#6#7#8\@{#9}{%
3287   \ifcase\@car#8\@nil\or    % Currently <10000, but prepared for bigger
3288     \bbl@alphnumerat@ii{#9}000000#1\or
3289     \bbl@alphnumerat@ii{#9}000000#1#2\or
3290     \bbl@alphnumerat@ii{#9}000000#1#2#3\or
3291     \bbl@alphnumerat@ii{#9}000000#1#2#3#4\else
3292     \bbl@alphnum@invalid{>9999}%
3293   \fi}
3294 \def\bbl@alphnumerat@ii#1#2#3#4#5#6#7#8{%
3295   \bbl@ifunset{bbl@cnt@#1.F.\@number#5#6#7#8\@language}%
3296     {\bbl@cs{cnt@#1.4\@language}{#5}
3297      \bbl@cs{cnt@#1.3\@language}{#6}
3298      \bbl@cs{cnt@#1.2\@language}{#7}
3299      \bbl@cs{cnt@#1.1\@language}{#8}
3300      \ifnum#6#7#8>\z@
3301        \bbl@ifunset{bbl@cnt@#1.S.321\@language}{}%
3302        {\bbl@cs{cnt@#1.S.321\@language}{}}
3303      \fi}%
3304   {\bbl@cs{cnt@#1.F.\@number#5#6#7#8\@language}}}
3305 \def\bbl@alphnum@invalid#1{%
3306   \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```

3307 \newcommand\BabelUppercaseMapping[3]{%
3308   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3309 \newcommand\BabelTitlecaseMapping[3]{%
3310   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3311 \newcommand\BabelLowercaseMapping[3]{%
3312   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

```

The parser for casing and casing.<variant>.

```

3313 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3314   \def\bbl@uftocode#1{\the\@numexpr\decode@UTFviii#1\relax}
3315 \else
3316   \def\bbl@uftocode#1{\expandafter`\string#1}
3317 \fi
3318 \def\bbl@casemapping#1#2#3{% 1:variant
3319   \def\bbl@tempa##1 ##2{% Loop
3320     \bbl@casemapping@i{##1}%
3321     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3322   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3323   \def\bbl@tempe{0}% Mode (upper/lower...)
3324   \def\bbl@tempc{#3}% Casing list
3325   \expandafter\bbl@tempa\bbl@tempc\@empty}
3326 \def\bbl@casemapping@i#1{%
3327   \def\bbl@tempb{#1}%
3328   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3329     \@nameuse{regex_replace_all:nnN}%
3330     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\0}}\bbl@tempb
3331   \else
3332     \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb % TODO. needed?
3333   \fi
3334   \expandafter\bbl@casemapping@ii\bbl@tempb\@}

```

```

3335 \def\bbl@casemapping@ii#1#2#3\@{%
3336   \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3337   \ifin@
3338     \edef\bbl@tempe{%
3339       \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3340   \else
3341     \ifcase\bbl@tempe\relax
3342       \DeclareUppercaseMapping[\bbl@templ]{\bbl@utfocode{#1}}{#2}%
3343       \DeclareLowercaseMapping[\bbl@templ]{\bbl@utfocode{#2}}{#1}%
3344     \or
3345       \DeclareUppercaseMapping[\bbl@templ]{\bbl@utfocode{#1}}{#2}%
3346     \or
3347       \DeclareLowercaseMapping[\bbl@templ]{\bbl@utfocode{#1}}{#2}%
3348     \or
3349       \DeclareTitlecaseMapping[\bbl@templ]{\bbl@utfocode{#1}}{#2}%
3350     \fi
3351   \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.

```

3352 \def\bbl@localeinfo#1#2{%
3353   \bbl@ifunset{bbl@info@#2}{#1}%
3354   {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}{#1}%
3355    {\bbl@cs{csname bbl@info@#2\endcsname @\language}}}%
3356 \newcommand\localeinfo[1]{%
3357   \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3358     \bbl@afterelse\bbl@localeinfo{}%
3359   \else
3360     \bbl@localeinfo
3361     {\bbl@error{no-ini-info}{}}{}%
3362     {#1}%
3363   \fi}
3364 % \@namedef{bbl@info@name.locale}{lcname}
3365 \@namedef{bbl@info@tag.ini}{lini}
3366 \@namedef{bbl@info@name.english}{elname}
3367 \@namedef{bbl@info@name.opentype}{lname}
3368 \@namedef{bbl@info@tag.bcp47}{tbc}
3369 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3370 \@namedef{bbl@info@tag.opentype}{lotf}
3371 \@namedef{bbl@info@script.name}{esname}
3372 \@namedef{bbl@info@script.name.opentype}{sname}
3373 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3374 \@namedef{bbl@info@script.tag.opentype}{sotf}
3375 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3376 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3377 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3378 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3379 \@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.

```

3380 << *More package options >> ≡
3381 \DeclareOption{ensureinfo=off}{}
3382 << /More package options >>
3383 \let\BabelEnsureInfo\relax

```

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

```

3384 \newcommand\getlocaleproperty{%
3385   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3386 \def\bbl@getproperty@s#1#2#3{%
3387   \let#1\relax
3388   \def\bbl@elt##1##2##3{%

```

```

3389 \bbl@ifsamestring{##1/##2}{#3}%
3390 {\providecommand#1{##3}%
3391 \def\bbl@elt###1###2###3{}}%
3392 {}}%
3393 \bbl@cs{inidata@#2}}%
3394 \def\bbl@getproperty@x#1#2#3{%
3395 \bbl@getproperty@#1}{#2}{#3}%
3396 \ifx#1\relax
3397 \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3398 \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`.

```

3399 \let\bbl@ini@loaded\@empty
3400 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3401 \def\ShowLocaleProperties#1{%
3402 \typeout{}}%
3403 \typeout{*** Properties for language '#1' ***}
3404 \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3405 \@nameuse{\bbl@inidata@#1}%
3406 \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.

```

3407 \newif\ifbbl@bcppallowed
3408 \bbl@bcppallowedfalse
3409 \def\bbl@autoload@options{import}
3410 \def\bbl@provide@locale{%
3411 \ifx\babelprovide\undefined
3412 \bbl@error{base-on-the-fly}{}{}%
3413 \fi
3414 \let\bbl@auxname\language % Still necessary. %^A TODO
3415 \ifbbl@bcptoname
3416 \bbl@ifunset{\bbl@bcp@map@\language}{}% Move uplevel??
3417 {\edef\language{\@nameuse{\bbl@bcp@map@\language}}}%
3418 \let\localname\language}%
3419 \fi
3420 \ifbbl@bcppallowed
3421 \expandafter\ifx\csname date\language\endcsname\relax
3422 \expandafter
3423 \bbl@bcplookup\language-\@empty-\@empty-\@empty@@
3424 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3425 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3426 \let\localname\language
3427 \expandafter\ifx\csname date\language\endcsname\relax
3428 \let\bbl@initoload\bbl@bcp
3429 \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3430 \let\bbl@initoload\relax
3431 \fi
3432 \bbl@csarg\xdef{\bcp@map@\bbl@bcp}{\localname}%
3433 \fi
3434 \fi
3435 \fi
3436 \expandafter\ifx\csname date\language\endcsname\relax
3437 \IfFileExists{babel-\language.tex}%
3438 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%

```

```

3439     {}%
3440   \fi}

```

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

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

```

3441 \providecommand\BCPdata{}
3442 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3443   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3444   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3445     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3446     {\bbl@bcpdata@ii#6}\bbl@main@language}%
3447     {\bbl@bcpdata@ii#1#2#3#4#5#6}\languagename}}%
3448   \def\bbl@bcpdata@ii#1#2{%
3449     \bbl@ifunset{\bbl@info@#1.tag.bcp47}%
3450     {\bbl@error{unknown-ini-field}{#1}}{}}%
3451     {\bbl@ifunset{\bbl@csname\bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3452     {\bbl@cs{\csname\bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3453 \fi
3454 \@namedef{\bbl@info@casing.tag.bcp47}{casing}
3455 \@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.

```

3456 \newcommand\babeladjust[1]{% TODO. Error handling.
3457   \bbl@forkv{#1}{%
3458     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3459     {\bbl@cs{ADJ@##1}{##2}}%
3460     {\bbl@cs{ADJ@##1@##2}}}
3461 %
3462 \def\bbl@adjust@lua#1#2{%
3463   \ifvmode
3464     \ifnum\currentgrouplevel=\z@
3465       \directlua{ Babel.#2 }%
3466       \expandafter\expandafter\expandafter\@gobble
3467     \fi
3468   \fi
3469   {\bbl@error{adjust-only-vertical}{#1}}{}}% Gobbled if everything went ok.
3470 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3471   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3472 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3473   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3474 \@namedef{\bbl@ADJ@bidi.text@on}{%
3475   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3476 \@namedef{\bbl@ADJ@bidi.text@off}{%
3477   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3478 \@namedef{\bbl@ADJ@bidi.math@on}{%
3479   \let\bbl@noamsmath\@empty}
3480 \@namedef{\bbl@ADJ@bidi.math@off}{%
3481   \let\bbl@noamsmath\relax}
3482 %
3483 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3484   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3485 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3486   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3487 %
3488 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3489   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3490 \@namedef{\bbl@ADJ@linebreak.sea@off}{%

```

```

3491 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3492 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3493 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3494 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3495 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3496 \@namedef{bbl@ADJ@justify.arabic@on}{%
3497 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3498 \@namedef{bbl@ADJ@justify.arabic@off}{%
3499 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3500 %
3501 \def\bbl@adjust@layout#1{%
3502 \ifvmode
3503 #1%
3504 \expandafter\@gobble
3505 \fi
3506 {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3507 \@namedef{bbl@ADJ@layout.tabular@on}{%
3508 \ifnum\bbl@tabular@mode=\tw@
3509 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3510 \else
3511 \chardef\bbl@tabular@mode\@ne
3512 \fi}
3513 \@namedef{bbl@ADJ@layout.tabular@off}{%
3514 \ifnum\bbl@tabular@mode=\tw@
3515 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3516 \else
3517 \chardef\bbl@tabular@mode\z@
3518 \fi}
3519 \@namedef{bbl@ADJ@layout.lists@on}{%
3520 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3521 \@namedef{bbl@ADJ@layout.lists@off}{%
3522 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3523 %
3524 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3525 \bbl@bcpallowedtrue}
3526 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3527 \bbl@bcpallowedfalse}
3528 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3529 \def\bbl@bcp@prefix{#1}}
3530 \def\bbl@bcp@prefix{bcp47-}
3531 \@namedef{bbl@ADJ@autoload.options}#1{%
3532 \def\bbl@autoload@options{#1}}
3533 \def\bbl@autoload@bcptoptions{import}
3534 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3535 \def\bbl@autoload@bcptoptions{#1}}
3536 \newif\ifbbl@bcptname
3537 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3538 \bbl@bcptnametrue}
3539 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3540 \bbl@bcptnamefalse}
3541 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3542 \directlua{ Babel.ignore_pre_char = function(node)
3543 return (node.lang == \the\csname l@nohyphenation\endcsname)
3544 end }}
3545 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3546 \directlua{ Babel.ignore_pre_char = function(node)
3547 return false
3548 end }}
3549 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3550 \def\bbl@ignoreinterchar{%
3551 \ifnum\language=\l@nohyphenation
3552 \expandafter\@gobble
3553 \else

```

```

3554 \expandafter\@firstofone
3555 \fi}}
3556 \@namedef{bbl@ADJ@interchar.disable@off}{%
3557 \let\bbl@ignoreinterchar\@firstofone}
3558 \@namedef{bbl@ADJ@select.write@shift}{%
3559 \let\bbl@restorelastskip\relax
3560 \def\bbl@savelastskip{%
3561 \let\bbl@restorelastskip\relax
3562 \ifvmode
3563 \ifdim\lastskip=\z@
3564 \let\bbl@restorelastskip\nobreak
3565 \else
3566 \bbl@exp{%
3567 \def\\bbl@restorelastskip{%
3568 \skip@=\the\lastskip
3569 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3570 \fi
3571 \fi}}
3572 \@namedef{bbl@ADJ@select.write@keep}{%
3573 \let\bbl@restorelastskip\relax
3574 \let\bbl@savelastskip\relax}
3575 \@namedef{bbl@ADJ@select.write@omit}{%
3576 \AddBabelHook{babel-select}{beforestart}{%
3577 \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3578 \let\bbl@restorelastskip\relax
3579 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3580 \@namedef{bbl@ADJ@select.encoding@off}{%
3581 \let\bbl@encoding@select@off\@empty}

```

5.1. Cross referencing macros

The \LaTeX book states:

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

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

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

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

```

3582 << *More package options >> ≡
3583 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3584 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3585 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3586 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3587 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3588 << /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.

```

3589 \bbl@trace{Cross referencing macros}
3590 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3591 \def\@newl@bel#1#2#3{%
3592 {\@safe@activetrue
3593 \bbl@ifunset{#1@#2}%
3594 \relax
3595 {\gdef\@multiplelabels{%
3596 \@latex@warning@no@line{There were multiply-defined labels}}%
3597 \@latex@warning@no@line{Label `#2' multiply defined}}%
3598 \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.

```

3599 \CheckCommand*\@testdef[3]{%
3600   \def\reserved@a{#3}%
3601   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3602   \else
3603     \@tempswatrue
3604   \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.

```

3605 \def\@testdef#1#2#3{% TODO. With @samestring?
3606   \@safe@activetrue
3607   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3608   \def\bbl@tempb{#3}%
3609   \@safe@activetrue
3610   \ifx\bbl@tempa\relax
3611   \else
3612     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3613   \fi
3614   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3615   \ifx\bbl@tempa\bbl@tempb
3616   \else
3617     \@tempswatrue
3618   \fi}
3619 \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.

```

3620 \bbl@xin@{R}\bbl@opt@safe
3621 \ifin@
3622   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3623   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3624   {\expandafter\strip@prefix\meaning\ref}%
3625 \ifin@
3626   \bbl@redefine\@kernel@ref#1{%
3627     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3628   \bbl@redefine\@kernel@pageref#1{%
3629     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3630   \bbl@redefine\@kernel@sref#1{%
3631     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3632   \bbl@redefine\@kernel@spageref#1{%
3633     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3634 \else
3635   \bbl@redefineroobust\ref#1{%
3636     \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3637   \bbl@redefineroobust\pageref#1{%
3638     \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3639 \fi
3640 \else
3641   \let\org@ref\ref
3642   \let\org@pageref\pageref
3643 \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.

```
3644 \bbl@xin@{B}\bbl@opt@safe
3645 \ifin@
3646 \bbl@redefine\@citex[#1]#2{%
3647   \@safe@activestrue\edef\bbl@tempa{#2}\@safe@activesfalse
3648   \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.)

```
3649 \AtBeginDocument{%
3650   \@ifpackageloaded{natbib}{%
3651     \def\@citex[#1][#2]#3{%
3652       \@safe@activestrue\edef\bbl@tempa{#3}\@safe@activesfalse
3653       \org@@citex[#1][#2]{\bbl@tempa}}%
3654   }}
```

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

```
3655 \AtBeginDocument{%
3656   \@ifpackageloaded{cite}{%
3657     \def\@citex[#1]#2{%
3658       \@safe@activestrue\org@@citex[#1][#2]\@safe@activesfalse}%
3659   }}
```

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

```
3660 \bbl@redefine\nocite#1{%
3661   \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3662 \bbl@redefine\bibcite{%
3663   \bbl@cite@choice
3664   \bibcite}
```

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

```
3665 \def\bbl@bibcite#1#2{%
3666   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3667 \def\bbl@cite@choice{%
3668   \global\let\bibcite\bbl@bibcite
3669   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3670   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3671   \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.

```
3672 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3673 \bbl@redefine\@bibitem#1{%
3674   \@safe@activestruelorg@@bibitem{#1}\@safe@activesfalse}
3675 \else
3676   \let\org@nocite\nocite
3677   \let\org@@citex\@citex
3678   \let\org@babcite\babcite
3679   \let\org@@bibitem\@bibitem
3680 \fi
```

5.2. Layout

```
3681 \newcommand\BabelPatchSection[1]{%
3682   \@ifundefined{#1}{}{%
3683     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3684     \@namedef{#1}{%
3685       \@ifstar{\bbl@presec@s{#1}}%
3686       {\@dblarg{\bbl@presec@x{#1}}}}}%
3687 \def\bbl@presec@x#1[#2]#3{%
3688   \bbl@exp{%
3689     \\select@language@x{\bbl@main@language}%
3690     \\bbl@cs{sspre@#1}%
3691     \\bbl@cs{ss@#1}%
3692     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3693     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3694     \\select@language@x{\language}}}%
3695 \def\bbl@presec@s#1#2{%
3696   \bbl@exp{%
3697     \\select@language@x{\bbl@main@language}%
3698     \\bbl@cs{sspre@#1}%
3699     \\bbl@cs{ss@#1}*%
3700     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3701     \\select@language@x{\language}}}%
3702 \IfBabelLayout{sectioning}%
3703   {\BabelPatchSection{part}%
3704    \BabelPatchSection{chapter}%
3705    \BabelPatchSection{section}%
3706    \BabelPatchSection{subsection}%
3707    \BabelPatchSection{subsubsection}%
3708    \BabelPatchSection{paragraph}%
3709    \BabelPatchSection{subparagraph}%
3710    \def\babel@toc#1{%
3711      \select@language@x{\bbl@main@language}}}%
3712 \IfBabelLayout{captions}%
3713   {\BabelPatchSection{caption}}}
```

5.3. Marks

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

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

```
3714 \bbl@trace{Marks}
3715 \IfBabelLayout{sectioning}
3716   {\ifx\bbl@opt@headfoot\@nnil
```

```

3717 \g@addto@macro\@resetactivechars{%
3718 \set@typeset@protect
3719 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3720 \let\protect\noexpand
3721 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3722 \edef\thepage{%
3723 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3724 \fi}%
3725 \fi}
3726 {\ifbbl@single\else
3727 \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3728 \markright#1{%
3729 \bbl@ifblank{#1}%
3730 {\org@markright{}}}%
3731 {\toks@{#1}%
3732 \bbl@exp{%
3733 \\\org@markright{\\protect\\foreignlanguage{\language}%
3734 {\\\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.)

```

3735 \ifx\@mkboth\markboth
3736 \def\bbl@tempc{\let\@mkboth\markboth}%
3737 \else
3738 \def\bbl@tempc{}%
3739 \fi
3740 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3741 \markboth#1#2{%
3742 \protected@edef\bbl@tempb##1{%
3743 \protect\foreignlanguage
3744 {\language}\{\protect\bbl@restore@actives##1}}%
3745 \bbl@ifblank{#1}%
3746 {\toks@{}}%
3747 {\toks@\expandafter{\bbl@tempb{#1}}}%
3748 \bbl@ifblank{#2}%
3749 {\@temptokena{}}%
3750 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3751 \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3752 \bbl@tempc
3753 \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.

```

3754 \bbl@trace{Preventing clashes with other packages}
3755 \ifx\org@ref\@undefined\else
3756   \bbl@xin@{R}\bbl@opt@safe
3757   \ifin@
3758     \AtBeginDocument{%
3759       \@ifpackageloaded{ifthen}{%
3760         \bbl@redefine@long\ifthenelse#1#2#3{%
3761           \let\bbl@temp@pref\pageref
3762           \let\pageref\org@pageref
3763           \let\bbl@temp@ref\ref
3764           \let\ref\org@ref
3765           \@safe@activestrue
3766           \org@ifthenelse{#1}%
3767             {\let\pageref\bbl@temp@pref
3768              \let\ref\bbl@temp@ref
3769              \@safe@activesfalse
3770              #2}%
3771             {\let\pageref\bbl@temp@pref
3772              \let\ref\bbl@temp@ref
3773              \@safe@activesfalse
3774              #3}%
3775           }%
3776         }{}%
3777       }
3778 \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`.

```

3779 \AtBeginDocument{%
3780   \@ifpackageloaded{varioref}{%
3781     \bbl@redefine\@@vpageref#1[#2]#3{%
3782       \@safe@activestrue
3783       \org@@vpageref{#1}[#2]#3}%
3784     \@safe@activesfalse}%
3785   \bbl@redefine\vrefpagemum#1#2{%
3786     \@safe@activestrue
3787     \org\vrefpagemum{#1}#2}%
3788   \@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.

```

3789   \expandafter\def\csname Ref \endcsname#1{%
3790     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3791   }{}%
3792 }
3793 \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.

```
3794 \AtEndOfPackage{%
3795   \AtBeginDocument{%
3796     \@ifpackageloaded{hhline}%
3797       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3798         \else
3799           \makeatletter
3800           \def\@currname{hhline}\input{hhline.sty}\makeatother
3801           \fi}%
3802     {}}}
```

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

```
3803 \def\substitutefontfamily#1#2#3{%
3804   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3805   \immediate\writel5{%
3806     \string\ProvidesFile{#1#2.fd}%
3807     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3808     \space generated font description file]^J
3809     \string\DeclareFontFamily{#1}{#2}{}}^J
3810     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3811     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3812     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3813     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3814     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3815     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3816     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3817     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3818   }%
3819   \closeout15
3820 }
3821 \@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

```
3822 \bbl@trace{Encoding and fonts}
3823 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3824 \newcommand\BabelNonText{TS1,T3,TS3}
3825 \let\org@TeX\TeX
3826 \let\org@LaTeX\LaTeX
3827 \let\ensureascii\@firstofone
3828 \let\asciientcoding\@empty
3829 \AtBeginDocument{%
3830   \def\@elt#1{,#1,}%
3831   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3832   \let\@elt\relax
3833   \let\bbl@tempb\@empty
3834   \def\bbl@tempc{OT1}%

```

```

3835 \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3836 \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3837 \bbl@foreach\bbl@tempa{%
3838 \bbl@xin@{, #1,}{, \BabelNonASCII,}%
3839 \ifin@
3840 \def\bbl@tempb{#1}% Store last non-ascii
3841 \else\bbl@xin@{, #1,}{, \BabelNonText,}% Pass
3842 \ifin@else
3843 \def\bbl@tempc{#1}% Store last ascii
3844 \fi
3845 \fi}%
3846 \ifx\bbl@tempb\@empty\else
3847 \bbl@xin@{, \cf@encoding,}{, \BabelNonASCII, \BabelNonText,}%
3848 \ifin@else
3849 \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3850 \fi
3851 \let\asciencoding\bbl@tempc
3852 \renewcommand\ensureascii[1]{%
3853 {\fontencoding{\asciencoding}\selectfont#1}}}%
3854 \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3855 \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3856 \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.

```

3857 \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.

```

3858 \AtBeginDocument{%
3859 \ifpackageloaded{fontspec}%
3860 {\xdef\latinencoding{%
3861 \ifx\UTFencname\undefined
3862 EU\ifcase\bbl@engine\or2\or1\fi
3863 \else
3864 \UTFencname
3865 \fi}}%
3866 {\gdef\latinencoding{OT1}%
3867 \ifx\cf@encoding\bbl@t@one
3868 \xdef\latinencoding{\bbl@t@one}%
3869 \else
3870 \def\elt#1{, #1,}%
3871 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3872 \let\elt\relax
3873 \bbl@xin@{, T1,}\bbl@tempa
3874 \ifin@
3875 \xdef\latinencoding{\bbl@t@one}%
3876 \fi
3877 \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.

```

3878 \DeclareRobustCommand{\latintext}{%
3879 \fontencoding{\latinencoding}\selectfont
3880 \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.

```

3881 \ifx\@undefined\DeclareTextFontCommand
3882   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3883 \else
3884   \DeclareTextFontCommand{\textlatin}{\latintext}
3885 \fi

```

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

```

3886 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.6. Basic bidi support

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

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

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

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

```

3887 \bbl@trace{Loading basic (internal) bidi support}
3888 \ifodd\bbl@engine
3889 \else % TODO. Move to txtbabel. Any xe+lua bidi
3890   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3891     \bbl@error{bidi-only-lua}{}}}%
3892   \let\bbl@beforeforeign\leavevmode
3893   \AtEndOfPackage{%
3894     \EnableBabelHook{babel-bidi}%
3895     \bbl@xebidipar}
3896 \fi\fi
3897 \def\bbl@loadxebidi#1{%
3898   \ifx\RTLfootnotetext\@undefined
3899     \AtEndOfPackage{%
3900       \EnableBabelHook{babel-bidi}%
3901       \ifx\fontspec\@undefined
3902         \usepackage{fontspec}% bidi needs fontspec
3903       \fi
3904       \usepackage#1{bidi}%
3905       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3906       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3907         \ifnum\@nameuse{\bbl@wdir@languagename}=\tw@ % 'AL' bidi
3908           \bbl@digitsdotdash % So ignore in 'R' bidi
3909         \fi}}%
3910   \fi}
3911 \ifnum\bbl@bidimode>200 % Any xe bidi=
3912   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3913     \bbl@tentative{bidi=bidi}
3914     \bbl@loadxebidi{}

```



```

3915 \or
3916 \bbl@loadxebidi{[rldocument]}
3917 \or
3918 \bbl@loadxebidi{}
3919 \fi
3920 \fi
3921 \fi
3922 % TODO? Separate:
3923 \ifnum\bbl@bidimode=\@ne % bidi=default
3924 \let\bbl@beforeforeign\leavevmode
3925 \ifodd\bbl@engine % lua
3926 \newattribute\bbl@attr@dir
3927 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3928 \bbl@exp{\output{\bodydir\pagedir\the\output}}
3929 \fi
3930 \AtEndOfPackage{%
3931 \EnableBabelHook{babel-bidi}% pdf/lua/x
3932 \ifodd\bbl@engine\else % pdf/x
3933 \bbl@xebidipar
3934 \fi}
3935 \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.

```

3936 \bbl@trace{Macros to switch the text direction}
3937 \def\bbl@alscripts{%
3938 ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
3939 \def\bbl@rscripts{%
3940 Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
3941 Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
3942 Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
3943 Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
3944 Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
3945 Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
3946 Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
3947 Meroitic,N'Ko,Orkhon,Todhri}
3948 \def\bbl@provide@dirs#1{%
3949 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3950 \ifin@
3951 \global\bbl@csarg\chardef{wdir@#1}\@ne
3952 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3953 \ifin@
3954 \global\bbl@csarg\chardef{wdir@#1}\tw@
3955 \fi
3956 \else
3957 \global\bbl@csarg\chardef{wdir@#1}\z@
3958 \fi
3959 \ifodd\bbl@engine
3960 \bbl@csarg\ifcase{wdir@#1}%
3961 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
3962 \or
3963 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
3964 \or
3965 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
3966 \fi
3967 \fi}
3968 \def\bbl@switchdir{%
3969 \bbl@ifunset\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3970 \bbl@ifunset\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3971 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
3972 \def\bbl@setdirs#1{% TODO - math
3973 \ifcase\bbl@select@type % TODO - strictly, not the right test

```

```

3974 \bbl@bodydir{#1}%
3975 \bbl@pardir{#1}% <- Must precede \bbl@textdir
3976 \fi
3977 \bbl@textdir{#1}}
3978 \ifnum\bbl@bidimode>\z@
3979 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3980 \DisableBabelHook{babel-bidi}
3981 \fi

```

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

```

3982 \ifodd\bbl@engine % luatex=1
3983 \else % pdftex=0, xetex=2
3984 \newcount\bbl@dirlevel
3985 \chardef\bbl@thetextdir\z@
3986 \chardef\bbl@thepardir\z@
3987 \def\bbl@textdir#1{%
3988 \ifcase#1\relax
3989 \chardef\bbl@thetextdir\z@
3990 \@nameuse{setlatin}%
3991 \bbl@textdir@i\beginL\endL
3992 \else
3993 \chardef\bbl@thetextdir\@ne
3994 \@nameuse{setnonlatin}%
3995 \bbl@textdir@i\beginR\endR
3996 \fi}
3997 \def\bbl@textdir@i#1#2{%
3998 \ifhmode
3999 \ifnum\currentgrouplevel>\z@
4000 \ifnum\currentgrouplevel=\bbl@dirlevel
4001 \bbl@error{multiple-bidi}{}{}{}%
4002 \bgroup\aftergroup#2\aftergroup\egroup
4003 \else
4004 \ifcase\currentgrouptype\or % 0 bottom
4005 \aftergroup#2% 1 simple {}
4006 \or
4007 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4008 \or
4009 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4010 \or\or\or % vbox vtop align
4011 \or
4012 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4013 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4014 \or
4015 \aftergroup#2% 14 \begingroup
4016 \else
4017 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4018 \fi
4019 \fi
4020 \bbl@dirlevel\currentgrouplevel
4021 \fi
4022 #1%
4023 \fi}
4024 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4025 \let\bbl@bodydir\@gobble
4026 \let\bbl@pagedir\@gobble
4027 \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).

```

4028 \def\bbl@xebidipar{%
4029 \let\bbl@xebidipar\relax
4030 \TeXeTstate\@ne
4031 \def\bbl@xeeverypar{%

```

```

4032 \ifcase\bb@thepardir
4033 \ifcase\bb@thetextdir\else\beginR\fi
4034 \else
4035 {\setbox\z@\lastbox\beginR\box\z@}%
4036 \fi}%
4037 \AddToHook{para/begin}{\bb@xeeverypar}}
4038 \ifnum\bb@bidimode>200 % Any xe bidi=
4039 \let\bb@textdir@i\@gobbletwo
4040 \let\bb@xebidipar\@empty
4041 \AddBabelHook{bidi}{foreign}{%
4042 \ifcase\bb@thetextdir
4043 \BabelWrapText{\LR{##1}}%
4044 \else
4045 \BabelWrapText{\RL{##1}}%
4046 \fi}
4047 \def\bb@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4048 \fi
4049 \fi

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

4050 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bb@textdir\z@#1}}
4051 \AtBeginDocument{%
4052 \ifx\pdfstringdefDisableCommands\@undefined\else
4053 \ifx\pdfstringdefDisableCommands\relax\else
4054 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4055 \fi
4056 \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.

```

4057 \bb@trace{Local Language Configuration}
4058 \ifx\loadlocalcfg\@undefined
4059 \@ifpackagewith{babel}{noconfigs}%
4060 {\let\loadlocalcfg\@gobble}%
4061 {\def\loadlocalcfg#1{%
4062 \InputIfFileExists{#1.cfg}%
4063 {\typeout{*****^J%
4064 * Local config file #1.cfg used^^J%
4065 *}}%
4066 \@empty}}
4067 \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).

```

4068 \bb@trace{Language options}
4069 \let\bb@afterlang\relax
4070 \let\BabelModifiers\relax
4071 \let\bb@loaded\@empty
4072 \def\bb@load@language#1{%
4073 \InputIfFileExists{#1.ldf}%
4074 {\edef\bb@loaded{CurrentOption
4075 \ifx\bb@loaded\@empty\else,\bb@loaded\fi}%
4076 \expandafter\let\expandafter\bb@afterlang

```

```

4077 \csname\CurrentOption.ldf-h@k\endcsname
4078 \expandafter\let\expandafter\BabelModifiers
4079 \csname bbl@mod@\CurrentOption\endcsname
4080 \bbl@exp{\AtBeginDocument{%
4081 \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4082 {\IfFileExists{babel-#1.tex}%
4083 {\def\bbl@tempa{%
4084 .\There is a locale ini file for this language.\%
4085 If it's the main language, try adding `provide=*'\%
4086 to the babel package options}}%
4087 {\let\bbl@tempa\empty}%
4088 \bbl@error{unknown-package-option}{}}}}

```

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

```

4089 \def\bbl@try@load@lang#1#2#3{%
4090 \IfFileExists{\CurrentOption.ldf}%
4091 {\bbl@load@language{\CurrentOption}}%
4092 {#1\bbl@load@language{#2}#3}}
4093 %
4094 \DeclareOption{friulian}{\bbl@try@load@lang}{\friulan}}
4095 \DeclareOption{hebrew}{%
4096 \ifcase\bbl@engine\or
4097 \bbl@error{only-pdfTeX-lang}{hebrew}{\LaTeX}}%
4098 \fi
4099 \input{rlbabel.def}%
4100 \bbl@load@language{hebrew}}
4101 \DeclareOption{hungarian}{\bbl@try@load@lang}{\magyar}}
4102 \DeclareOption{lowersorbian}{\bbl@try@load@lang}{\lsorbian}}
4103 % \DeclareOption{northernkurdish}{\bbl@try@load@lang}{\kurmanji}}
4104 \DeclareOption{polutonikogreek}{%
4105 \bbl@try@load@lang}{\greek}{\languageattribute{greek}{polutoniko}}}
4106 \DeclareOption{russian}{\bbl@try@load@lang}{\russianb}}
4107 \DeclareOption{ukrainian}{\bbl@try@load@lang}{\ukraineb}}
4108 \DeclareOption{uppersorbian}{\bbl@try@load@lang}{\usorbian}}

```

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

If the language has been set as metadata, read the info from the corresponding ini file and extract the babel name. Then added it 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 main option we set here explicitly.

```

4109 \ifx\GetDocumentProperties\undefined\else
4110 \edef\bbl@metalang{\GetDocumentProperties{document/lang}}%
4111 \ifx\bbl@metalang\empty\else
4112 \expandafter
4113 \bbl@bcpllookup\bbl@metalang-\@empty-\@empty-\@empty@@
4114 {\bbl@read@ini{\bbl@bcp}\m@ne}%
4115 \edef\bbl@language@opts{\bbl@language@opts,\language}%
4116 \ifx\bbl@opt@main\@nnil
4117 \let\bbl@opt@main\language
4118 \fi
4119 \bbl@info{Passing \language space to babel}%
4120 \fi
4121 \fi
4122 \ifx\bbl@opt@config\@nnil
4123 \@ifpackagewith{babel}{noconfigs}}%
4124 {\InputIfFileExists{bblopts.cfg}%
4125 {\typeout{*****^J%
4126 * Local config file bblopts.cfg used^^J%
4127 *}}}%

```

```

4128     {}}%
4129 \else
4130   \InputIfFileExists{\bbl@opt@config.cfg}%
4131   {\typeout{*****^J%
4132     * Local config file \bbl@opt@config.cfg used^J%
4133     *}}%
4134   {\bbl@error{config-not-found}}{}{}%
4135 \fi

```

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

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

```

4136 \def\bbl@tempf{,}
4137 \bbl@foreach\@raw@classoptionslist{%
4138   \in@{=}{#1}%
4139   \ifin@else
4140     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty}%
4141   \fi}
4142 \ifx\bbl@opt@main\@nnil
4143   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4144     \let\bbl@tempb\@empty
4145     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4146     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4147     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4148       \ifx\bbl@opt@main\@nnil % i.e., if not yet assigned
4149         \ifodd\bbl@iniflag % = *=
4150           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4151         \else % n +=
4152           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4153         \fi
4154       \fi}%
4155   \fi
4156 \else
4157   \ifx\bbl@metalang\@undefined\else\ifx\bbl@metalang\@empty\else
4158     \bbl@afterfi\expandafter\@gobble
4159   \fi\fi % except if explicit lang metatag:
4160   {\bbl@info{Main language set with 'main='. Except if you have\\%
4161     problems, prefer the default mechanism for setting\\%
4162     the main language, i.e., as the last declared.\\%
4163     Reported}}
4164 \fi

```

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

```

4165 \ifx\bbl@opt@main\@nnil\else
4166   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4167   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4168 \fi

```

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

```

4169 \bbl@foreach\bbl@language@opts{%
4170   \def\bbl@tempa{#1}%
4171   \ifx\bbl@tempa\bbl@opt@main\else
4172     \ifnum\bbl@iniflag<\tw@ % 0 ∅ (other = ldf)
4173       \bbl@ifunset{ds@#1}%
4174       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4175     {}%

```

```

4176     \else                                % + * (other = ini)
4177     \DeclareOption{#1}{%
4178     \bbl@ldfinit
4179     \babelprovide[@import]{#1}% %%%
4180     \bbl@afterldf}%
4181     \fi
4182 \fi}
4183 \bbl@foreach\bbl@tempf{%
4184 \def\bbl@tempa{#1}%
4185 \ifx\bbl@tempa\bbl@opt@main\else
4186 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4187 \bbl@ifunset{ds@#1}%
4188 {\IfFileExists{#1.ldf}%
4189 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4190 {}}%
4191 {}}%
4192 \else                                % + * (other = ini)
4193 \IfFileExists{babel-#1.tex}%
4194 {\DeclareOption{#1}{%
4195 \bbl@ldfinit
4196 \babelprovide[@import]{#1}% %%%
4197 \bbl@afterldf}}%
4198 {}}%
4199 \fi
4200 \fi}

```

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

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

```

4201 \NewHook{babel/presets}
4202 \UseHook{babel/presets}
4203 \def\AfterBabelLanguage#1{%
4204 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}}
4205 \DeclareOption*{}
4206 \ProcessOptions*

```

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

```

4207 \bbl@trace{Option 'main'}
4208 \ifx\bbl@opt@main@nnil
4209 \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4210 \let\bbl@tempc@empty
4211 \edef\bbl@templ{\bbl@loaded,}
4212 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4213 \bbl@for\bbl@tempb\bbl@tempa{%
4214 \edef\bbl@tempd{\bbl@tempb,}%
4215 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4216 \bbl@xin{\bbl@tempd}{\bbl@templ}%
4217 \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4218 \def\bbl@tempa#1,#2@nnil{\def\bbl@tempb{#1}}
4219 \expandafter\bbl@tempa\bbl@loaded,@nnil
4220 \ifx\bbl@tempb\bbl@tempc\else
4221 \bbl@warning{%
4222 Last declared language option is '\bbl@tempc',\%
4223 but the last processed one was '\bbl@tempb'.\%
4224 The main language can't be set as both a global\%
4225 and a package option. Use 'main=\bbl@tempc' as\%
4226 option. Reported}

```

```

4227 \fi
4228 \else
4229 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4230 \bbl@ldfinit
4231 \let\CurrentOption\bbl@opt@main
4232 \bbl@exp{% \bbl@opt@provide = empty if *
4233 \\\bbl@provide
4234 [\bbl@opt@provide,@import,main]% %%%
4235 {\bbl@opt@main}}%
4236 \bbl@afterldf
4237 \DeclareOption{\bbl@opt@main}{}
4238 \else % case 0,2 (main is ldf)
4239 \ifx\bbl@loadmain\relax
4240 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4241 \else
4242 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4243 \fi
4244 \ExecuteOptions{\bbl@opt@main}
4245 \@namedef{ds@\bbl@opt@main}{}%
4246 \fi
4247 \DeclareOption*{}
4248 \ProcessOptions*
4249 \fi
4250 \bbl@exp{%
4251 \\\AtBeginDocument{\\\bbl@usehooks@lang/{\begin{document}}{}}}%
4252 \def\AfterBabelLanguage{\bbl@error{late-after-babel}}{}{}

```

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

```

4253 \ifx\bbl@main@language\undefined
4254 \bbl@info{%
4255   You haven't specified a language as a class or package\\%
4256   option. I'll load 'nil'. Reported}
4257 \bbl@load@language{nil}
4258 \fi
4259 \end{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`

```

4260 \begin{kernel}
4261 \let\bbl@onlyswitch\empty
4262 \input babel.def
4263 \let\bbl@onlyswitch\undefined
4264 \end{kernel}

```

7. Error messages

They are loaded when `\bbl@error` is first called. To save space, the main code just identifies them with a tag, and messages are stored in a separate file. Since it can be loaded anywhere, you make

sure some catcodes have the right value, although those for \, `, ^M, % and = are reset before loading the file.

```

4265 <*errors>
4266 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4267 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4268 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4269 \catcode`\@=11 \catcode`\^=7
4270 %
4271 \ifx\MessageBreak\undefined
4272 \gdef\bbl@error@i#1#2{%
4273     \beginingroup
4274     \newlinechar=`^^J
4275     \def\{^J(babel) }%
4276     \errhelp{#2}\errmessage{\{#1}%
4277     \endgroup}
4278 \else
4279 \gdef\bbl@error@i#1#2{%
4280     \beginingroup
4281     \def\{\MessageBreak}%
4282     \PackageError{babel}{#1}{#2}%
4283     \endgroup}
4284 \fi
4285 \def\bbl@errmessage#1#2#3{%
4286 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4287     \bbl@error@i{#2}{#3}}
4288 % Implicit #2#3#4:
4289 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4290 %
4291 \bbl@errmessage{not-yet-available}
4292     {Not yet available}%
4293     {Find an armchair, sit down and wait}
4294 \bbl@errmessage{bad-package-option}%
4295     {Bad option '#1=#2'. Either you have misspelled the\\%
4296     key or there is a previous setting of '#1'. Valid\\%
4297     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4298     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4299     {See the manual for further details.}
4300 \bbl@errmessage{base-on-the-fly}
4301     {For a language to be defined on the fly 'base'\\%
4302     is not enough, and the whole package must be\\%
4303     loaded. Either delete the 'base' option or\\%
4304     request the languages explicitly}%
4305     {See the manual for further details.}
4306 \bbl@errmessage{undefined-language}
4307     {You haven't defined the language '#1' yet.\\%
4308     Perhaps you misspelled it or your installation\\%
4309     is not complete}%
4310     {Your command will be ignored, type <return> to proceed}
4311 \bbl@errmessage{shorthand-is-off}
4312     {I can't declare a shorthand turned off (\string#2)}
4313     {Sorry, but you can't use shorthands which have been\\%
4314     turned off in the package options}
4315 \bbl@errmessage{not-a-shorthand}
4316     {The character '\string #1' should be made a shorthand character;\\%
4317     add the command \string\usesshorthands\string{#1\string} to
4318     the preamble.\\%
4319     I will ignore your instruction}%
4320     {You may proceed, but expect unexpected results}
4321 \bbl@errmessage{not-a-shorthand-b}
4322     {I can't switch '\string#2' on or off--not a shorthand}%
4323     {This character is not a shorthand. Maybe you made\\%
4324     a typing mistake? I will ignore your instruction.}
4325 \bbl@errmessage{unknown-attribute}

```



```

4326 {The attribute #2 is unknown for language #1.}%
4327 {Your command will be ignored, type <return> to proceed}
4328 \bbl@errmessage{missing-group}
4329 {Missing group for string \string#1}%
4330 {You must assign strings to some category, typically\\%
4331 captions or extras, but you set none}
4332 \bbl@errmessage{only-lua-xe}
4333 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4334 {Consider switching to these engines.}
4335 \bbl@errmessage{only-lua}
4336 {This macro is available only in LuaLaTeX}%
4337 {Consider switching to that engine.}
4338 \bbl@errmessage{unknown-provide-key}
4339 {Unknown key '#1' in \string\babelprovide}%
4340 {See the manual for valid keys}%
4341 \bbl@errmessage{unknown-mapfont}
4342 {Option '\bbl@KVP@mapfont' unknown for\\%
4343 mapfont. Use 'direction'}%
4344 {See the manual for details.}
4345 \bbl@errmessage{no-ini-file}
4346 {There is no ini file for the requested language\\%
4347 (#1: \language). Perhaps you misspelled it or your\\%
4348 installation is not complete}%
4349 {Fix the name or reinstall babel.}
4350 \bbl@errmessage{digits-is-reserved}
4351 {The counter name 'digits' is reserved for mapping\\%
4352 decimal digits}%
4353 {Use another name.}
4354 \bbl@errmessage{limit-two-digits}
4355 {Currently two-digit years are restricted to the\\
4356 range 0-9999}%
4357 {There is little you can do. Sorry.}
4358 \bbl@errmessage{alphabetic-too-large}
4359 {Alphabetic numeral too large (#1)}%
4360 {Currently this is the limit.}
4361 \bbl@errmessage{no-ini-info}
4362 {I've found no info for the current locale.\\%
4363 The corresponding ini file has not been loaded\\%
4364 Perhaps it doesn't exist}%
4365 {See the manual for details.}
4366 \bbl@errmessage{unknown-ini-field}
4367 {Unknown field '#1' in \string\BCPdata.\\%
4368 Perhaps you misspelled it}%
4369 {See the manual for details.}
4370 \bbl@errmessage{unknown-locale-key}
4371 {Unknown key for locale '#2':\\%
4372 #3\\%
4373 \string#1 will be set to \string\relax}%
4374 {Perhaps you misspelled it.}%
4375 \bbl@errmessage{adjust-only-vertical}
4376 {Currently, #1 related features can be adjusted only\\%
4377 in the main vertical list}%
4378 {Maybe things change in the future, but this is what it is.}
4379 \bbl@errmessage{layout-only-vertical}
4380 {Currently, layout related features can be adjusted only\\%
4381 in vertical mode}%
4382 {Maybe things change in the future, but this is what it is.}
4383 \bbl@errmessage{bidi-only-lua}
4384 {The bidi method 'basic' is available only in\\%
4385 luatex. I'll continue with 'bidi=default', so\\%
4386 expect wrong results}%
4387 {See the manual for further details.}
4388 \bbl@errmessage{multiple-bidi}

```

```

4389 {Multiple bidi settings inside a group}%
4390 {I'll insert a new group, but expect wrong results.}
4391 \bbl@errmessage{unknown-package-option}
4392 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4393 or the language definition file \CurrentOption.ldf\\%
4394 was not found%
4395 \bbl@tempa}
4396 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4397 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4398 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4399 \bbl@errmessage{config-not-found}
4400 {Local config file '\bbl@opt@config.cfg' not found}%
4401 {Perhaps you misspelled it.}
4402 \bbl@errmessage{late-after-babel}
4403 {Too late for \string\AfterBabelLanguage}%
4404 {Languages have been loaded, so I can do nothing}
4405 \bbl@errmessage{double-hyphens-class}
4406 {Double hyphens aren't allowed in \string\babelcharclass\\%
4407 because it's potentially ambiguous}%
4408 {See the manual for further info}
4409 \bbl@errmessage{unknown-interchar}
4410 {'#1' for '\language' cannot be enabled.\\%
4411 Maybe there is a typo}%
4412 {See the manual for further details.}
4413 \bbl@errmessage{unknown-interchar-b}
4414 {'#1' for '\language' cannot be disabled.\\%
4415 Maybe there is a typo}%
4416 {See the manual for further details.}
4417 \bbl@errmessage{charproperty-only-vertical}
4418 {\string\babelcharproperty\space can be used only in\\%
4419 vertical mode (preamble or between paragraphs)}%
4420 {See the manual for further info}
4421 \bbl@errmessage{unknown-char-property}
4422 {No property named '#2'. Allowed values are\\%
4423 direction (bc), mirror (bmg), and linebreak (lb)}%
4424 {See the manual for further info}
4425 \bbl@errmessage{bad-transform-option}
4426 {Bad option '#1' in a transform.\\%
4427 I'll ignore it but expect more errors}%
4428 {See the manual for further info.}
4429 \bbl@errmessage{font-conflict-transforms}
4430 {Transforms cannot be re-assigned to different\\%
4431 fonts. The conflict is in '\bbl@kv@label'.\\%
4432 Apply the same fonts or use a different label}%
4433 {See the manual for further details.}
4434 \bbl@errmessage{transform-not-available}
4435 {'#1' for '\language' cannot be enabled.\\%
4436 Maybe there is a typo or it's a font-dependent transform}%
4437 {See the manual for further details.}
4438 \bbl@errmessage{transform-not-available-b}
4439 {'#1' for '\language' cannot be disabled.\\%
4440 Maybe there is a typo or it's a font-dependent transform}%
4441 {See the manual for further details.}
4442 \bbl@errmessage{year-out-range}
4443 {Year out of range.\\%
4444 The allowed range is #1}%
4445 {See the manual for further details.}
4446 \bbl@errmessage{only-pdfTeX-lang}
4447 {The '#1' ldf style doesn't work with #2,\\%
4448 but you can use the ini locale instead.\\%
4449 Try adding 'provide=' to the option list. You may\\%
4450 also want to set 'bidi=' to some value}%
4451 {See the manual for further details.}

```

```

4452 \bbl@errmessage{hyphenmins-args}
4453 {\string\babelhyphenmins\ accepts either the optional\%
4454 argument or the star, but not both at the same time}%
4455 {See the manual for further details.}
4456 </errors>
4457 <*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.

```

4458 <@Make sure ProvidesFile is defined@>
4459 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4460 \xdef\bbl@format{\jobname}
4461 \def\bbl@version{<@version@>}
4462 \def\bbl@date{<@date@>}
4463 \ifx\AtBeginDocument\undefined
4464 \def\@empty{}
4465 \fi
4466 <@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.

```

4467 \def\process@line#1#2 #3 #4 {%
4468   \ifx=#1%
4469     \process@synonym{#2}%
4470   \else
4471     \process@language{#1#2}{#3}{#4}%
4472   \fi
4473   \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.

```

4474 \toks@{}
4475 \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.

```

4476 \def\process@synonym#1{%
4477   \ifnum\last@language=\m@ne
4478     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4479   \else
4480     \expandafter\chardef\curname l@#1\endcurname\last@language
4481     \wlog{\string\l@#1=\string\language\the\last@language}%
4482     \expandafter\let\curname #1hyphenmins\expandafter\endcurname
4483     \curname\language\hyphenmins\endcurname
4484     \let\bbl@elt\relax
4485     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4486   \fi}

```

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

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

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

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

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

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

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

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

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

```

4487 \def\process@language#1#2#3{%
4488   \expandafter\addlanguage\csname l@#1\endcsname
4489   \expandafter\language\csname l@#1\endcsname
4490   \edef\languagename{#1}%
4491   \bbl@hook@everylanguage{#1}%
4492   % > luatex
4493   \bbl@get@enc#1:.\@@@
4494   \begingroup
4495     \lefthyphenmin\m@ne
4496     \bbl@hook@loadpatterns{#2}%
4497     % > luatex
4498     \ifnum\lefthyphenmin=\m@ne
4499       \else
4500         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4501           \the\lefthyphenmin\the\righthyphenmin}%
4502         \fi
4503     \endgroup
4504     \def\bbl@tempa{#3}%
4505     \ifx\bbl@tempa\@empty\else
4506       \bbl@hook@loadexceptions{#3}%
4507       % > luatex
4508     \fi
4509     \let\bbl@elt\relax
4510     \edef\bbl@languages{%
4511       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4512     \ifnum\the\language=\z@
4513       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4514         \set@hyphenmins\tw@\thr@\relax
4515       \else
4516         \expandafter\expandafter\expandafter\set@hyphenmins
4517         \csname #1hyphenmins\endcsname
4518       \fi
4519       \the\toks@
4520       \toks@{}%
4521     \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.

```

4522 \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.

```

4523 \def\bbl@hook@everylanguage#1{}
4524 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4525 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4526 \def\bbl@hook@loadkernel#1{%
4527   \def\addlanguage{\csname newlanguage\endcsname}%
4528   \def\adddialect##1##2{%
4529     \global\chardef##1##2\relax
4530     \wlog{\string##1 = a dialect from \string\language##2}}%
4531   \def\iflanguage##1{%
4532     \expandafter\ifx\csname l@##1\endcsname\relax
4533       \nolater{##1}%
4534     \else
4535       \ifnum\csname l@##1\endcsname=\language
4536         \expandafter\expandafter\expandafter\@firstoftwo
4537       \else
4538         \expandafter\expandafter\expandafter\@secondoftwo
4539       \fi
4540     \fi}%
4541   \def\providehyphenmins##1##2{%
4542     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4543       \namedef{##1hyphenmins}{##2}%
4544     \fi}%
4545   \def\set@hyphenmins##1##2{%
4546     \lefthyphenmin##1\relax
4547     \righthyphenmin##2\relax}%
4548   \def\selectlanguage{%
4549     \errhelp{Selecting a language requires a package supporting it}%
4550     \errmessage{No multilingual package has been loaded}}%
4551   \let\foreignlanguage\selectlanguage
4552   \let\otherlanguage\selectlanguage
4553   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4554   \def\bbl@usehooks##1##2{% TODO. Temporary!!
4555   \def\setlocale{%
4556     \errhelp{Find an armchair, sit down and wait}%
4557     \errmessage{(babel) Not yet available}}%
4558   \let\uselocale\setlocale
4559   \let\locale\setlocale
4560   \let\selectlocale\setlocale
4561   \let\localename\setlocale
4562   \let\textlocale\setlocale
4563   \let\textlanguage\setlocale
4564   \let\languagetext\setlocale}
4565 \begingroup
4566 \def\AddBabelHook#1#2{%
4567   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4568     \def\next{\toks1}%
4569   \else
4570     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4571   \fi
4572   \next}
4573 \ifx\directlua\@undefined
4574   \ifx\XeTeXinputencoding\@undefined\else
4575     \input xebabel.def
4576   \fi
4577 \else
4578   \input luababel.def
4579 \fi
4580 \openin1 = babel-\bbl@format.cfg
4581 \ifeof1
4582 \else

```

```

4583 \input babel-\bbl@format.cfg\relax
4584 \fi
4585 \closein1
4586 \endgroup
4587 \bbl@hook@loadkernel{switch.def}

```

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

```

4588 \openin1 = language.dat

```

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

```

4589 \def\language{english}%
4590 \ifeof1
4591 \message{I couldn't find the file language.dat,\space
4592         I will try the file hyphen.tex}
4593 \input hyphen.tex\relax
4594 \chardef\l@english\z@
4595 \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`.

```

4596 \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.

```

4597 \loop
4598 \endlinechar\m@ne
4599 \read1 to \bbl@line
4600 \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.

```

4601 \if T\ifeof1\fi T\relax
4602 \ifx\bbl@line\empty\else
4603 \edef\bbl@line{\bbl@line\space\space\space}%
4604 \expandafter\process@line\bbl@line\relax
4605 \fi
4606 \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.

```

4607 \begingroup
4608 \def\bbl@elt#1#2#3#4{%
4609 \global\language=#2\relax
4610 \gdef\language{#1}%
4611 \def\bbl@elt##1##2##3##4{}}%
4612 \bbl@languages
4613 \endgroup
4614 \fi
4615 \closein1

```

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

```

4616 \if/\the\toks@\else
4617 \errhelp{language.dat loads no language, only synonyms}
4618 \errmessage{Orphan language synonym}
4619 \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.

```

4620 \let\bbl@line\@undefined
4621 \let\process@line\@undefined
4622 \let\process@synonym\@undefined
4623 \let\process@language\@undefined
4624 \let\bbl@get@enc\@undefined
4625 \let\bbl@hyph@enc\@undefined
4626 \let\bbl@tempa\@undefined
4627 \let\bbl@hook@loadkernel\@undefined
4628 \let\bbl@hook@everylanguage\@undefined
4629 \let\bbl@hook@loadpatterns\@undefined
4630 \let\bbl@hook@loadexceptions\@undefined
4631 </patterns>

```

Here the code for `iniTeX` ends.

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

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

```

4632 <<More package options>> ≡
4633 \chardef\bbl@bidimode\z@
4634 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4635 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4636 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4637 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4638 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4639 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4640 <</More package options>>

```

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

```

4641 <<Font selection>> ≡
4642 \bbl@trace{Font handling with fontspec}
4643 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4644 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckckstdfonts}
4645 \DisableBabelHook{babel-fontspec}
4646 \@onlypreamble\babelfont
4647 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4648   \ifx\fontspec\undefined
4649     \usepackage{fontspec}%
4650     \fi
4651     \EnableBabelHook{babel-fontspec}%
4652     \edef\bbl@tempa{#1}%
4653     \def\bbl@tempb{#2}% Used by \bbl@bblfont
4654     \bbl@bblfont}
4655 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4656   \bbl@ifunset{\bbl@tempb family}%
4657     {\bbl@providefam{\bbl@tempb}}%
4658     {}%
4659   % For the default font, just in case:
4660   \bbl@ifunset{\bbl@lsys\language}{\bbl@provide\lsys{\language}}{}%
4661   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4662     {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save \bbl@rmdflt@
4663     \bbl@exp{%
4664       \let<\bbl@tempb dflt@\language>\<\bbl@tempb dflt@>%
4665       \\bbl@font@set<\bbl@tempb dflt@\language>%
4666       <\bbl@tempb default>\<\bbl@tempb family>}}%

```

```

4667 {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *srt
4668 \bbl@csarg\def{\bbl@tempb dflt@##1}{<#1>{#1}{#2}}}}}%

```

If the family in the previous command does not exist, it must be defined. Here is how:

```

4669 \def\bbl@providfam#1{%
4670 \bbl@exp{%
4671 \\\newcommand\<#1default>{}% Just define it
4672 \\\bbl@add@list\\bbl@font@fams{#1}%
4673 \\\NewHook{#1family}%
4674 \\\DeclareRobustCommand\<#1family>{%
4675 \\\not@math@alphabet\<#1family>\relax
4676 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4677 \\\fontfamily\<#1default>%
4678 \\\UseHook{#1family}%
4679 \\\selectfont}%
4680 \\\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.

```

4681 \def\bbl@nostdfont#1{%
4682 \bbl@ifunset{\bbl@WFF@f@family}%
4683 {\bbl@csarg\gdef{WFF@f@family}{}% Flag, to avoid dupl warns
4684 \bbl@infowarn{The current font is not a babel standard family:\%
4685 #1%
4686 \fontname\font\\%
4687 There is nothing intrinsically wrong with this warning, and\\%
4688 you can ignore it altogether if you do not need these\\%
4689 families. But if they are used in the document, you should be\\%
4690 aware 'babel' will not set Script and Language for them, so\\%
4691 you may consider defining a new family with \string\babelfont.\\%
4692 See the manual for further details about \string\babelfont.\\%
4693 Reported}}
4694 {}}%
4695 \gdef\bbl@switchfont{%
4696 \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys{language}}}%
4697 \bbl@exp{% e.g., Arabic -> arabic
4698 \lowercase{\edef\\bbl@tempa{\bbl@ccl{sname}}}}}%
4699 \bbl@foreach\bbl@font@fams{%
4700 \bbl@ifunset{\bbl@##1dflt@language}% (1) language?
4701 {\bbl@ifunset{\bbl@##1dflt@*bbl@tempa}% (2) from script?
4702 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4703 {}% 123=F - nothing!
4704 {\bbl@exp{% 3=T - from generic
4705 \global\let\bbl@##1dflt@language>%
4706 \<bbl@##1dflt@>}}}%
4707 {\bbl@exp{% 2=T - from script
4708 \global\let\bbl@##1dflt@language>%
4709 \<bbl@##1dflt@*bbl@tempa>}}}%
4710 {}}% 1=T - language, already defined
4711 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4712 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4713 \bbl@ifunset{\bbl@##1dflt@language}%
4714 {\bbl@cs{famrst@##1}%
4715 \global\bbl@csarg\let{famrst@##1}\relax}%
4716 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4717 \\\bbl@add\\originalTeX%
4718 \\\bbl@font@rst{\bbl@ccl{##1dflt}}%
4719 \<##1default>\<##1family>{##1}}%
4720 \\\bbl@font@set{\<bbl@##1dflt@language>% the main part!
4721 \<##1default>\<##1family>}}}%
4722 \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.


```

4723 \ifx\f@family\undefined\else % if latex
4724 \ifcase\bbl@engine % if pdftex
4725 \let\bbl@cckcckstdfonts\relax
4726 \else
4727 \def\bbl@cckcckstdfonts{%
4728 \begingroup
4729 \global\let\bbl@cckcckstdfonts\relax
4730 \let\bbl@tempa\empty
4731 \bbl@foreach\bbl@font@fams{%
4732 \bbl@ifunset{\bbl@##1dflt@}%
4733 {\@nameuse{##1family}}%
4734 \bbl@csarg\gdef{WFF@f@family}}}% Flag
4735 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= \f@family\\}%
4736 \space\space\fontname\font\\}%
4737 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4738 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4739 {}}%
4740 \ifx\bbl@tempa\empty\else
4741 \bbl@infowarn{The following font families will use the default\\%
4742 settings for all or some languages:\\%
4743 \bbl@tempa
4744 There is nothing intrinsically wrong with it, but\\%
4745 'babel' will no set Script and Language, which could\\%
4746 be relevant in some languages. If your document uses\\%
4747 these families, consider redefining them with \string\babelfont.\\%
4748 Reported}%
4749 \fi
4750 \endgroup}
4751 \fi
4752 \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, \TeX 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*`).

```

4753 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4754 \bbl@xin@{<>}{#1}%
4755 \ifin@
4756 \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4757 \fi
4758 \bbl@exp{%
4759 \def\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
4760 \bbl@ifsamestring{#2}{\f@family}%
4761 {\#3%
4762 \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}}%
4763 \let\bbl@tempa\relax}%
4764 {}}}

```

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).

```

4765 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4766 \let\bbl@tempe\bbl@mapselect
4767 \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
4768 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}}%
4769 \let\bbl@mapselect\relax

```

```

4770 \let\bb@temp@fam#4%      e.g., '\rmfamily', to be restored below
4771 \let#4\@empty           %      Make sure \renewfontfamily is valid
4772 \bb@set@renderer
4773 \bb@exp{%
4774   \let\bb@temp@pfam\<\bb@stripslash#4\space>% e.g., '\rmfamily '
4775   \<keys_if_exist:nnF>{\fontspec-opentype}{Script/\bb@cl{sname}}}%
4776   {\newfontscript{\bb@cl{sname}}{\bb@cl{sotf}}}%
4777   \<keys_if_exist:nnF>{\fontspec-opentype}{Language/\bb@cl{lname}}}%
4778   {\newfontlanguage{\bb@cl{lname}}{\bb@cl{lotf}}}%
4779   \renewfontfamily\#4%
4780   [\bb@cl{sys},% xetex removes unknown features :-(
4781     \ifcase\bb@engine\or RawFeature={family=\bb@tempb},\fi
4782     #2]}{#3}% i.e., \bb@exp{.}{#3}
4783 \bb@unset@renderer
4784 \begingroup
4785   #4%
4786   \xdef#1{\f@family}%      e.g., \bb@rmdflt@lang{FreeSerif(0)}
4787 \endgroup % TODO. Find better tests:
4788 \bb@xin@{\string>\string s\string s\string u\string b\string*}%
4789   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4790 \ifin@
4791   \global\bb@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4792 \fi
4793 \bb@xin@{\string>\string s\string s\string u\string b\string*}%
4794   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4795 \ifin@
4796   \global\bb@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4797 \fi
4798 \let#4\bb@temp@fam
4799 \bb@exp{\let\<\bb@stripslash#4\space>}\bb@temp@pfam
4800 \let\bb@mapselect\bb@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.

```

4801 \def\bb@font@rst#1#2#3#4{%
4802   \bb@ccarg\def{famrst@#4}{\bb@font@set{#1}#2#3}}

```

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

```

4803 \def\bb@font@fams{rm,sf,tt}
4804 <</Font selection>>

```

\BabelFootnote Footnotes.

```

4805 <<*Footnote changes>> ≡
4806 \bb@trace{Bidi footnotes}
4807 \ifnum\bb@bidimode>\z@ % Any bidi=
4808   \def\bb@footnote#1#2#3{%
4809     \@ifnextchar[%
4810       {\bb@footnote@o{#1}{#2}{#3}}%
4811       {\bb@footnote@x{#1}{#2}{#3}}}
4812 \long\def\bb@footnote@x#1#2#3#4{%
4813   \bgroup
4814     \select@language{x{\bb@main@language}}%
4815     \bb@fn@footnote{#2#1{\ignorespaces#4}#3}%
4816   \egroup}
4817 \long\def\bb@footnote@o#1#2#3[#4]#5{%
4818   \bgroup
4819     \select@language{x{\bb@main@language}}%
4820     \bb@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4821   \egroup}
4822 \def\bb@footnotetext#1#2#3{%
4823   \@ifnextchar[%
4824     {\bb@footnotetext@o{#1}{#2}{#3}}%

```

```

4825     {\bbl@footnotetext@x{#1}{#2}{#3}}
4826 \long\def\bbl@footnotetext@x#1#2#3#4{%
4827     \bgroup
4828     \select@language@x{\bbl@main@language}%
4829     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4830     \egroup}
4831 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4832     \bgroup
4833     \select@language@x{\bbl@main@language}%
4834     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4835     \egroup}
4836 \def\BabelFootnote#1#2#3#4{%
4837     \ifx\bbl@fn@footnote\undefined
4838     \let\bbl@fn@footnote\footnote
4839     \fi
4840     \ifx\bbl@fn@footnotetext\undefined
4841     \let\bbl@fn@footnotetext\footnotetext
4842     \fi
4843     \bbl@ifblank{#2}%
4844     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4845     \namedef{\bbl@stripslash#1text}%
4846     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4847     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4848     \namedef{\bbl@stripslash#1text}%
4849     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4850 \fi
4851 <</Footnote changes>>

```

10. Hooks for XeTeX and LuaTeX

10.1. XeTeX

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

Now, the code.

```

4852 <*<xtex>
4853 \def\BabelStringsDefault{unicode}
4854 \let\xebbl@stop\relax
4855 \AddBabelHook{xetex}{encodedcommands}{%
4856     \def\bbl@tempa{#1}%
4857     \ifx\bbl@tempa\empty
4858     \XeTeXinputencoding"bytes"%
4859     \else
4860     \XeTeXinputencoding"#1"%
4861     \fi
4862     \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4863 \AddBabelHook{xetex}{stopcommands}{%
4864     \xebbl@stop
4865     \let\xebbl@stop\relax}
4866 \def\bbl@input@classes{% Used in CJK intraspaces
4867     \input{load-unicode-xetex-classes.tex}%
4868     \let\bbl@input@classes\relax}
4869 \def\bbl@intraspace#1 #2 #3\@@{%
4870     \bbl@csarg\gdef{xeisp@\languagename}%
4871     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4872 \def\bbl@intrapenalty#1\@@{%
4873     \bbl@csarg\gdef{xeipn@\languagename}%
4874     {\XeTeXlinebreakpenalty #1\relax}}
4875 \def\bbl@provide@intraspace{%
4876     \bbl@xin@{/s}{\bbl@cl{lnbrk}}}%
4877 \ifin@ \else \bbl@xin@{/c}{\bbl@cl{lnbrk}}\fi
4878 \ifin@

```

```

4879 \bbl@ifunset{bbl@intsp@{language}}{%
4880   {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\empty\else
4881     \ifx\bbl@KVP@intraspace\@nnil
4882       \bbl@exp{%
4883         \\bbl@intraspace\bbl@cl{intsp}\\\@}%
4884       \fi
4885       \ifx\bbl@KVP@intrapenalty\@nnil
4886         \bbl@intrapenalty0\@@
4887       \fi
4888     \fi
4889     \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4890       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4891     \fi
4892     \ifx\bbl@KVP@intrapenalty\@nnil\else
4893       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4894     \fi
4895     \bbl@exp{%
4896       % TODO. Execute only once (but redundant):
4897       \\bbl@add{<extras{language}>{%
4898         \XeTeXlinebreaklocale "\bbl@cl{tbc}}"%
4899         \<bbl@xeisp@{language}>%
4900         \<bbl@xeipn@{language}>%
4901         \\bbl@tglobal{<extras{language}>%
4902         \\bbl@add{<noextras{language}>{%
4903         \XeTeXlinebreaklocale ""}%
4904         \\bbl@tglobal{<noextras{language}>%
4905       \ifx\bbl@ispace\@undefined
4906         \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4907       \ifx\AtBeginDocument\@notprerr
4908         \expandafter\@secondoftwo % to execute right now
4909       \fi
4910       \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4911     \fi}%
4912 \fi}
4913 \ifx\DisableBabelHook\@undefined\endinput\fi %%% TODO: why
4914 \let\bbl@set@renderer\relax
4915 \let\bbl@unset@renderer\relax
4916 <@Font selection@>
4917 \def\bbl@provide@extra#1{}

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

4918 \def\bbl@xenohyph@{%
4919   \bbl@ifset{bbl@prehc@{language}}%
4920     {\ifnum\hyphenchar\font=\defaultthyphenchar
4921       \iffontchar\font\bbl@cl{prehc}\relax
4922       \hyphenchar\font\bbl@cl{prehc}\relax
4923     \else\iffontchar\font"200B
4924       \hyphenchar\font"200B
4925     \else
4926       \bbl@warning
4927         {Neither 0 nor ZERO WIDTH SPACE are available\\%
4928         in the current font, and therefore the hyphen\\%
4929         will be printed. Try changing the fontspec's\\%
4930         'HyphenChar' to another value, but be aware\\%
4931         this setting is not safe (see the manual).\\%
4932         Reported}%
4933       \hyphenchar\font\defaultthyphenchar
4934     \fi\fi
4935   \fi}%
4936 {\hyphenchar\font\defaultthyphenchar}}

```

10.2. Support for interchar

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

```
4937 \ifnum\xe@alloc@intercharclass<\thr@@
4938 \xe@alloc@intercharclass\thr@@
4939 \fi
4940 \chardef\bbl@xe@class@default@=\z@
4941 \chardef\bbl@xe@class@cjkideogram@=\@ne
4942 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
4943 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
4944 \chardef\bbl@xe@class@boundary@=4095
4945 \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.

```
4946 \AddBabelHook{babel-interchar}{beforeextras}{%
4947 \nameuse{bbl@xechars@\language@}}
4948 \DisableBabelHook{babel-interchar}
4949 \protected\def\bbl@charclass#1{%
4950 \ifnum\count@<\z@
4951 \count@-\count@
4952 \loop
4953 \bbl@exp{%
4954 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4955 \XeTeXcharclass\count@ \bbl@tempc
4956 \ifnum\count@<`#1\relax
4957 \advance\count@\@ne
4958 \repeat
4959 \else
4960 \babel@savevariable{\XeTeXcharclass`#1}%
4961 \XeTeXcharclass`#1 \bbl@tempc
4962 \fi
4963 \count@`#1\relax}
```

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

```
4964 \newcommand\bbl@ifinterchar[1]{%
4965 \let\bbl@tempa\@gobble % Assume to ignore
4966 \edef\bbl@tempb{\zap@space#1 \@empty}%
4967 \ifx\bbl@KVP@interchar\@nnil\else
4968 \bbl@replace\bbl@KVP@interchar{ }{,}%
4969 \bbl@foreach\bbl@tempb{%
4970 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
4971 \ifin@
4972 \let\bbl@tempa\@firstofone
4973 \fi}%
4974 \fi
4975 \bbl@tempa}
4976 \newcommand\IfBabelIntercharT[2]{%
4977 \bbl@carg\bbl@add{bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
4978 \newcommand\babelcharclass[3]{%
4979 \EnableBabelHook{babel-interchar}%
4980 \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
4981 \def\bbl@tempb##1{%
4982 \ifx##1\@empty\else
4983 \ifx##1-%
4984 \bbl@upto
```

```

4985 \else
4986 \bbl@charclass{%
4987 \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4988 \fi
4989 \expandafter\bbl@tempb
4990 \fi}%
4991 \bbl@ifunset{bbl@xechars@#1}%
4992 {\toks@{%
4993 \babel@savevariable\XeTeXinterchartokenstate
4994 \XeTeXinterchartokenstate@ne
4995 }}%
4996 {\toks@\expandafter\expandafter\expandafter{%
4997 \csname bbl@xechars@#1\endcsname}}%
4998 \bbl@csarg\edef{xechars@#1}{%
4999 \the\toks@
5000 \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
5001 \bbl@tempb#3\@empty}}
5002 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
5003 \protected\def\bbl@upto{%
5004 \ifnum\count@>\z@
5005 \advance\count@\@ne
5006 \count@-\count@
5007 \else\ifnum\count@=\z@
5008 \bbl@charclass{-}%
5009 \else
5010 \bbl@error{double-hyphens-class}{\count@}{\count@}%
5011 \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>`.

```

5012 \def\bbl@ignoreinterchar{%
5013 \ifnum\language=\l@nohyphenation
5014 \expandafter\@gobble
5015 \else
5016 \expandafter\@firstofone
5017 \fi}
5018 \newcommand\babelinterchar[5][]{%
5019 \let\bbl@kv@label\@empty
5020 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5021 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5022 {\bbl@ignoreinterchar{#5}}%
5023 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5024 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5025 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5026 \XeTeXinterchartoks
5027 \@nameuse{bbl@xeclasse@\bbl@tempa @#2}
5028 \bbl@ifunset{bbl@xeclasse@\bbl@tempa @#2}{\bbl@tempa @#2} %
5029 \@nameuse{bbl@xeclasse@\bbl@tempb @#2}
5030 \bbl@ifunset{bbl@xeclasse@\bbl@tempb @#2}{\bbl@tempb @#2} %
5031 = \expandafter{%
5032 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5033 \csname\zap@space bbl@xeinter@\bbl@kv@label
5034 @#3@#4@#2 \@empty\endcsname}}}}
5035 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5036 \bbl@ifunset{bbl@ic@#1@language}%
5037 {\bbl@error{unknown-interchar}{#1}{\count@}}%
5038 {\bbl@csarg\let{ic@#1@language}\@firstofone}}
5039 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5040 \bbl@ifunset{bbl@ic@#1@language}%
5041 {\bbl@error{unknown-interchar-b}{#1}{\count@}}%
5042 {\bbl@csarg\let{ic@#1@language}\@gobble}}
5043 </xetex>

```

10.3. Layout

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

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

Consider `txtbabel` as a shorthand for *tex-xet babel*, which is the bidi model in both pdf_{te}x and xet_{ex}.

```
5044 <*xetex | texxet>
5045 \providecommand\bbl@provide@intraspace{}
5046 \bbl@trace{Redefinitions for bidi layout}
5047 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5048 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5049 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5050 \ifnum\bbl@bidimode>\z@ % TODO: always?
5051   \def\hangfrom#1{%
5052     \setbox\@tempboxa\hbox{#1}%
5053     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5054     \noindent\box\@tempboxa}
5055 \def\raggedright{%
5056   \let\\\@centercr
5057   \bbl@startskip\z@skip
5058   \@rightskip\@flushglue
5059   \bbl@endskip\@rightskip
5060   \parindent\z@
5061   \parfillskip\bbl@startskip}
5062 \def\raggedleft{%
5063   \let\\\@centercr
5064   \bbl@startskip\@flushglue
5065   \bbl@endskip\z@skip
5066   \parindent\z@
5067   \parfillskip\bbl@endskip}
5068 \fi
5069 \IfBabelLayout{lists}
5070 {\bbl@sreplace\list
5071   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5072   \def\bbl@listleftmargin{%
5073     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5074   \ifcase\bbl@engine
5075     \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5076     \def\p@enumiii{\p@enumii}\theenumii}%
5077   \fi
5078   \bbl@sreplace\@verbatim
5079     {\leftskip\@totalleftmargin}%
5080     {\bbl@startskip\textwidth
5081       \advance\bbl@startskip-\linewidth}%
5082   \bbl@sreplace\@verbatim
5083     {\rightskip\z@skip}%
5084     {\bbl@endskip\z@skip}}%
5085 {}
5086 \IfBabelLayout{contents}
5087 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5088   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5089 {}
5090 \IfBabelLayout{columns}
5091 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outpuhbox}%
5092   \def\bbl@outpuhbox#1{%
5093     \hb@xt@\textwidth{%
5094       \hskip\columnwidth
5095       \hfil
5096       {\normalcolor\vrule \@width\columnseprule}%
5097       \hfil
```

```

5098      \hb@xt@{columnwidth}{\box\@leftcolumn \hss}%
5099      \hskip-\textwidth
5100      \hb@xt@{columnwidth}{\box\@outputbox \hss}%
5101      \hskip\columnsep
5102      \hskip\columnwidth}}}%
5103  {}
5104  <@Footnote changes@>
5105  \IfBabelLayout{footnotes}%
5106  {\BabelFootnote\footnote\language\language{}{}}%
5107   \BabelFootnote\localfootnote\language\language{}{}}%
5108   \BabelFootnote\mainfootnote{}{}}{}{}}
5109  {}

```

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

```

5110 \IfBabelLayout{counters*}%
5111  {\bbl@add\bbl@opt@layout{.counters.}%
5112   \AddToHook{shipout/before}{%
5113     \let\bbl@tempa\babelsublr
5114     \let\babelsublr\@firstofone
5115     \let\bbl@save@thepage\thepage
5116     \protected@edef\thepage{\thepage}%
5117     \let\babelsublr\bbl@tempa}%
5118   \AddToHook{shipout/after}{%
5119     \let\thepage\bbl@save@thepage}}{}
5120 \IfBabelLayout{counters}%
5121  {\let\bbl@latinarabic=\@arabic
5122   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
5123   \let\bbl@asciroman=\@roman
5124   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5125   \let\bbl@asciiRoman=\@Roman
5126   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5127 \fi % end if layout
5128 </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).

```

5129 < *texxet>
5130 \def\bbl@provide@extra#1{%
5131   % == auto-select encoding ==
5132   \ifx\bbl@encoding@select@off\@empty\else
5133     \bbl@ifunset{\bbl@encoding@#1}%
5134     {\def\@elt##1{,##1,}%
5135      \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5136      \count@z@
5137      \bbl@foreach\bbl@tempe{%
5138        \def\bbl@tempd{##1}% Save last declared
5139        \advance\count@\@ne}%
5140      \ifnum\count@>\@ne % (1)
5141        \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5142        \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5143        \bbl@replace\bbl@tempa{ },}%
5144        \global\bbl@csarg\let{encoding@#1}\@empty
5145        \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5146        \ifin@ \else % if main encoding included in ini, do nothing
5147          \let\bbl@tempb\relax
5148          \bbl@foreach\bbl@tempa{%
5149            \ifx\bbl@tempb\relax
5150              \bbl@xin@{,##1,}{,\bbl@tempe,}%
5151              \ifin@\def\bbl@tempb{##1}\fi
5152            \fi}%

```



```

5153         \ifx\bbbl@tempb\relax\else
5154         \bbbl@exp{%
5155             \global\<bbbl@add>\<bbbl@preextras@#1>\<bbbl@encoding@#1>}%
5156         \gdef\<bbbl@encoding@#1>{%
5157             \\babel@save\\f@encoding
5158             \\bbbl@add\\originalTeX{\\selectfont}%
5159             \\fontencoding{\bbbl@tempb}%
5160             \\selectfont}}%
5161     \fi
5162     \fi
5163     \fi}%
5164     {}%
5165     \fi}
5166     /<texet>

```

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, `\bbbl@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`).

```

5167     <*<luatex>
5168     \directlua{ Babel = Babel or {} } % DL2
5169     \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5170     \bbbl@trace{Read language.dat}
5171     \ifx\bbbl@readstream\undefined
5172         \csname newread\endcsname\bbbl@readstream
5173     \fi
5174     \begingroup
5175     \toks@{}
5176     \count@% % 0=start, 1=0th, 2=normal
5177     \def\bbbl@process@line#1#2 #3 #4 {%
5178         \ifx=#1%
5179             \bbbl@process@synonym{#2}%
5180         \else

```

```

5181     \bbl@process@language{#1#2}{#3}{#4}%
5182     \fi
5183     \ignorespaces}
5184 \def\bbl@manylang{%
5185     \ifnum\bbl@last>\@ne
5186         \bbl@info{Non-standard hyphenation setup}%
5187     \fi
5188     \let\bbl@manylang\relax}
5189 \def\bbl@process@language#1#2#3{%
5190     \ifcase\count@
5191         \ifundefined{zth@#1}{\count@tw@}{\count@ne}%
5192     \or
5193         \count@tw@
5194     \fi
5195     \ifnum\count@=tw@
5196         \expandafter\addlanguage\csname l@#1\endcsname
5197         \language\allocationnumber
5198         \chardef\bbl@last\allocationnumber
5199         \bbl@manylang
5200         \let\bbl@elt\relax
5201         \xdef\bbl@languages{%
5202             \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5203     \fi
5204     \the\toks@
5205     \toks@{}}
5206 \def\bbl@process@synonym@aux#1#2{%
5207     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5208     \let\bbl@elt\relax
5209     \xdef\bbl@languages{%
5210         \bbl@languages\bbl@elt{#1}{#2}{}}}%
5211 \def\bbl@process@synonym#1{%
5212     \ifcase\count@
5213         \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5214     \or
5215         \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{%
5216         \else
5217             \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5218         \fi}
5219 \ifx\bbl@languages@undefined % Just a (sensible?) guess
5220     \chardef\l@english\z@
5221     \chardef\l@USenglish\z@
5222     \chardef\bbl@last\z@
5223     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5224     \gdef\bbl@languages{%
5225         \bbl@elt{english}{0}{hyphen.tex}}%
5226     \bbl@elt{USenglish}{0}{}%
5227 \else
5228     \global\let\bbl@languages@format\bbl@languages
5229     \def\bbl@elt#1#2#3#4{% Remove all except language 0
5230         \ifnum#2>\z@
5231             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5232         \fi}%
5233     \xdef\bbl@languages{\bbl@languages}%
5234 \fi
5235 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5236 \bbl@languages
5237 \openin\bbl@readstream=language.dat
5238 \ifeof\bbl@readstream
5239     \bbl@warning{I couldn't find language.dat. No additional\\%
5240                 patterns loaded. Reported}%
5241 \else
5242     \loop
5243         \endlinechar\m@ne

```

```

5244 \read\bbl@readstream to \bbl@line
5245 \endlinechar\^^M
5246 \if T\ifeof\bbl@readstream F\fi T\relax
5247 \ifx\bbl@line\empty\else
5248 \edef\bbl@line{\bbl@line\space\space\space}%
5249 \expandafter\bbl@process@line\bbl@line\relax
5250 \fi
5251 \repeat
5252 \fi
5253 \closein\bbl@readstream
5254 \endgroup
5255 \bbl@trace{Macros for reading patterns files}
5256 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5257 \ifx\babelcatcodetablenum\undefined
5258 \ifx\newcatcodetable\undefined
5259 \def\babelcatcodetablenum{5211}
5260 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5261 \else
5262 \newcatcodetable\babelcatcodetablenum
5263 \newcatcodetable\bbl@pattcodes
5264 \fi
5265 \else
5266 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5267 \fi
5268 \def\bbl@luapatterns#1#2{%
5269 \bbl@get@enc#1:.\@@@
5270 \setbox\z@\hbox\bgroup
5271 \beginingroup
5272 \savecatcodetable\babelcatcodetablenum\relax
5273 \initcatcodetable\bbl@pattcodes\relax
5274 \catcodetable\bbl@pattcodes\relax
5275 \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5276 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5277 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5278 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5279 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5280 \catcode\`=12 \catcode\'=12 \catcode\"=12
5281 \input #1\relax
5282 \catcodetable\babelcatcodetablenum\relax
5283 \endgroup
5284 \def\bbl@tempa{#2}%
5285 \ifx\bbl@tempa\empty\else
5286 \input #2\relax
5287 \fi
5288 \egroup}%
5289 \def\bbl@patterns@lua#1{%
5290 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5291 \csname l@#1\endcsname
5292 \edef\bbl@tempa{#1}%
5293 \else
5294 \csname l@#1:\f@encoding\endcsname
5295 \edef\bbl@tempa{#1:\f@encoding}%
5296 \fi\relax
5297 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5298 \@ifundefined{bbl@hyphendata@the\language}%
5299 {\def\bbl@elt##1##2##3##4{%
5300 \ifnum##2=\csname l@#1:\f@encoding\endcsname % #2=spanish, dutch:OT1...
5301 \def\bbl@tempb{##3}%
5302 \ifx\bbl@tempb\empty\else % if not a synonymous
5303 \def\bbl@tempc{{##3}{##4}}%
5304 \fi
5305 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5306 \fi}%

```

```

5307 \bbl@languages
5308 \ifundefined{bbl@hyphendata@the\language}%
5309 {\bbl@info{No hyphenation patterns were set for\%
5310 language '\bbl@tempa'. Reported}}%
5311 {\expandafter\expandafter\expandafter\bbl@luapatterns
5312 \csname bbl@hyphendata@the\language\endcsname}}{}
5313 \endinput\fi

```

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

```

5314 \ifx\DisableBabelHook\@undefined
5315 \AddBabelHook{luatex}{everylanguage}{%
5316 \def\process@language##1##2##3{%
5317 \def\process@line####1####2 ####3 ####4 {}}
5318 \AddBabelHook{luatex}{loadpatterns}{%
5319 \input #1\relax
5320 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5321 {{#1}}}}
5322 \AddBabelHook{luatex}{loadexceptions}{%
5323 \input #1\relax
5324 \def\bbl@tempb##1##2{{##1}{#1}}%
5325 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5326 {\expandafter\expandafter\expandafter\bbl@tempb
5327 \csname bbl@hyphendata@the\language\endcsname}}
5328 \endinput\fi

```

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

```

5329 \beginingroup % TODO - to a lua file % DL3
5330 \catcode`\%=12
5331 \catcode`\'=12
5332 \catcode`\|=12
5333 \catcode`\:=12
5334 \directlua{
5335 Babel.locale_props = Babel.locale_props or {}
5336 function Babel.lua_error(e, a)
5337 tex.print([[noexpand\csname bbl@error\endcsname]] ..
5338 e .. '{' .. (a or '') .. '}{}{')
5339 end
5340 function Babel.bytes(line)
5341 return line:gsub("(.)",
5342 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5343 end
5344 function Babel.begin_process_input()
5345 if luatexbase and luatexbase.add_to_callback then
5346 luatexbase.add_to_callback('process_input_buffer',
5347 Babel.bytes, 'Babel.bytes')
5348 else
5349 Babel.callback = callback.find('process_input_buffer')
5350 callback.register('process_input_buffer', Babel.bytes)
5351 end
5352 end
5353 function Babel.end_process_input ()
5354 if luatexbase and luatexbase.remove_from_callback then
5355 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5356 else
5357 callback.unregister('process_input_buffer', Babel.callback)
5358 end
5359 end
5360 function Babel.str_to_nodes(fn, matches, base)
5361 local n, head, last
5362 if fn == nil then return nil end
5363 for s in string.utfvalues(fn(matches)) do
5364 if base.id == 7 then
5365 base = base.replace

```

```

5366     end
5367     n = node.copy(base)
5368     n.char = s
5369     if not head then
5370         head = n
5371     else
5372         last.next = n
5373     end
5374     last = n
5375 end
5376 return head
5377 end
5378 Babel.linebreaking = Babel.linebreaking or {}
5379 Babel.linebreaking.before = {}
5380 Babel.linebreaking.after = {}
5381 Babel.locale = {}
5382 function Babel.linebreaking.add_before(func, pos)
5383     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5384     if pos == nil then
5385         table.insert(Babel.linebreaking.before, func)
5386     else
5387         table.insert(Babel.linebreaking.before, pos, func)
5388     end
5389 end
5390 function Babel.linebreaking.add_after(func)
5391     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5392     table.insert(Babel.linebreaking.after, func)
5393 end
5394 function Babel.addpatterns(pp, lg)
5395     local lg = lang.new(lg)
5396     local pats = lang.patterns(lg) or ''
5397     lang.clear_patterns(lg)
5398     for p in pp:gmatch('[^%s]+') do
5399         ss = ''
5400         for i in string.utfcharacters(p:gsub('%d', '')) do
5401             ss = ss .. '%d?' .. i
5402         end
5403         ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5404         ss = ss:gsub('%.%d%?$', '%%.')
5405         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5406         if n == 0 then
5407             tex.sprint(
5408                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5409                 .. p .. [[]])
5410             pats = pats .. ' ' .. p
5411         else
5412             tex.sprint(
5413                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5414                 .. p .. [[]])
5415         end
5416     end
5417     lang.patterns(lg, pats)
5418 end
5419 Babel.characters = Babel.characters or {}
5420 Babel.ranges = Babel.ranges or {}
5421 function Babel.hlist_has_bidi(head)
5422     local has_bidi = false
5423     local ranges = Babel.ranges
5424     for item in node.traverse(head) do
5425         if item.id == node.id'glyph' then
5426             local itemchar = item.char
5427             local chardata = Babel.characters[itemchar]
5428             local dir = chardata and chardata.d or nil

```

```

5429         if not dir then
5430             for nn, et in ipairs(ranges) do
5431                 if itemchar < et[1] then
5432                     break
5433                 elseif itemchar <= et[2] then
5434                     dir = et[3]
5435                     break
5436                 end
5437             end
5438         end
5439         if dir and (dir == 'al' or dir == 'r') then
5440             has_bidi = true
5441         end
5442     end
5443 end
5444 return has_bidi
5445 end
5446 function Babel.set_chrnges_b (script, chrng)
5447     if chrng == '' then return end
5448     texio.write('Replacing ' .. script .. ' script ranges')
5449     Babel.script_blocks[script] = {}
5450     for s, e in string.gmatch(chrng..' ', '(.)%..(.)%s') do
5451         table.insert(
5452             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5453     end
5454 end
5455 function Babel.discard_sublr(str)
5456     if str:find( [[\string\indexentry]] ) and
5457        str:find( [[\string\babelsublr]] ) then
5458         str = str:gsub( [[\string\babelsublr%s*{%b{}}]],
5459                        function(m) return m:sub(2,-2) end )
5460     end
5461     return str
5462 end
5463 }
5464 \endgroup
5465 \ifx\newattribute\@undefined\else % Test for plain
5466     \newattribute\bbl@attr@locale % DL4
5467     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5468     \AddBabelHook{luatex}{beforeextras}{%
5469         \setattribute\bbl@attr@locale\localeid}
5470 \fi
5471 \def\BabelStringsDefault{unicode}
5472 \let\luabbl@stop\relax
5473 \AddBabelHook{luatex}{encodedcommands}{%
5474     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5475     \ifx\bbl@tempa\bbl@tempb\else
5476         \directlua{Babel.begin_process_input()}%
5477         \def\luabbl@stop{%
5478             \directlua{Babel.end_process_input()}}%
5479     \fi}%
5480 \AddBabelHook{luatex}{stopcommands}{%
5481     \luabbl@stop
5482     \let\luabbl@stop\relax}
5483 \AddBabelHook{luatex}{patterns}{%
5484     \@ifundefined{bbl@hyphendata@the\language}%
5485     {\def\bbl@elt##1##2##3##4{%
5486         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5487         \def\bbl@tempb{##3}%
5488         \ifx\bbl@tempb\empty\else % if not a synonymous
5489             \def\bbl@tempc{{##3}{##4}}%
5490         \fi
5491         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%

```

```

5492     \fi}%
5493     \bbl@languages
5494     \ifundefined{bbl@hyphendata@the\language}%
5495     {\bbl@info{No hyphenation patterns were set for\%
5496       language '#2'. Reported}}%
5497     {\expandafter\expandafter\expandafter\bbl@luapatterns
5498       \csname bbl@hyphendata@the\language\endcsname}}}%
5499     \ifundefined{bbl@patterns@}\fi}%
5500     \begingroup
5501     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5502     \ifin@else
5503     \ifx\bbl@patterns@\empty\else
5504     \directlua{ Babel.addpatterns(
5505       [[\bbl@patterns@]], \number\language) }%
5506     \fi
5507     \ifundefined{bbl@patterns@#1}%
5508     \@empty
5509     {\directlua{ Babel.addpatterns(
5510       [[\space\csname bbl@patterns@#1\endcsname]],
5511       \number\language) }}%
5512     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5513     \fi
5514     \endgroup}%
5515     \bbl@exp{%
5516     \bbl@ifunset{bbl@prehc@languagename}{}%
5517     {\bbl@ifblank{\bbl@cs{prehc@languagename}}}%
5518     {\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.

```

5519 \onlypreamble\babelpatterns
5520 \AtEndOfPackage{%
5521   \newcommand\babelpatterns[2][\empty]{%
5522     \ifx\bbl@patterns@\relax
5523       \let\bbl@patterns@\empty
5524     \fi
5525     \ifx\bbl@pttnlist@\empty\else
5526     \bbl@warning{%
5527       You must not intermingle \string\selectlanguage\space and\%
5528       \string\babelpatterns\space or some patterns will not\%
5529       be taken into account. Reported}%
5530     \fi
5531     \ifx@\empty#1%
5532     \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5533     \else
5534     \edef\bbl@tempb{\zap@space#1 \empty}%
5535     \bbl@for\bbl@tempa\bbl@tempb{%
5536       \bbl@fixname\bbl@tempa
5537       \bbl@iflanguage\bbl@tempa{%
5538         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5539           \ifundefined{bbl@patterns@\bbl@tempa}%
5540             \empty
5541             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5542             #2}}}%
5543     \fi}}

```

10.6. Southeast Asian scripts

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

Replace regular (i.e., implicit) discretionary spaces by spaceskip, 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.

```

5544 \def\bbl@intraspace#1 #2 #3\@{%
5545   \directlua{
5546     Babel.intraspaces = Babel.intraspaces or {}
5547     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5548       {b = #1, p = #2, m = #3}
5549     Babel.locale_props[\the\localeid].intraspace = %
5550       {b = #1, p = #2, m = #3}
5551   }}
5552 \def\bbl@intrapenalty#1\@{%
5553   \directlua{
5554     Babel.intrapenalties = Babel.intrapenalties or {}
5555     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5556     Babel.locale_props[\the\localeid].intrapenalty = #1
5557   }}
5558 \begingroup
5559 \catcode`\%=12
5560 \catcode`\&=14
5561 \catcode`\'=12
5562 \catcode`\-=12
5563 \gdef\bbl@seaintraspace&
5564   \let\bbl@seaintraspace\relax
5565   \directlua{
5566     Babel.sea_enabled = true
5567     Babel.sea_ranges = Babel.sea_ranges or {}
5568     function Babel.set_chranges (script, chrng)
5569       local c = 0
5570       for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5571         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5572         c = c + 1
5573       end
5574     end
5575     function Babel.sea_disc_to_space (head)
5576       local sea_ranges = Babel.sea_ranges
5577       local last_char = nil
5578       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5579       for item in node.traverse(head) do
5580         local i = item.id
5581         if i == node.id'glyph' then
5582           last_char = item
5583         elseif i == 7 and item.subtype == 3 and last_char
5584           and last_char.char > 0x0C99 then
5585           quad = font.getfont(last_char.font).size
5586           for lg, rg in pairs(sea_ranges) do
5587             if last_char.char > rg[1] and last_char.char < rg[2] then
5588               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5589               local intraspace = Babel.intraspaces[lg]
5590               local intrapenalty = Babel.intrapenalties[lg]
5591               local n
5592               if intrapenalty ~= 0 then
5593                 n = node.new(14, 0)      &% penalty
5594                 n.penalty = intrapenalty
5595                 node.insert_before(head, item, n)
5596               end
5597               n = node.new(12, 13)      &% (glue, spaceskip)
5598               node.setglue(n, intraspace.b * quad,
5599                 intraspace.p * quad,
5600                 intraspace.m * quad)
5601               node.insert_before(head, item, n)
5602               node.remove(head, item)
5603             end
5604           end
5605         end

```



```

5606     end
5607   end
5608 }&
5609 \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.

```

5610 \catcode`\%=14
5611 \gdef\bbl@cjkintraspacespace{%
5612   \let\bbl@cjkintraspacespace\relax
5613   \directlua{
5614     require('babel-data-cjk.lua')
5615     Babel.cjk_enabled = true
5616     function Babel.cjk_linebreak(head)
5617       local GLYPH = node.id'glyph'
5618       local last_char = nil
5619       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5620       local last_class = nil
5621       local last_lang = nil
5622       for item in node.traverse(head) do
5623         if item.id == GLYPH then
5624           local lang = item.lang
5625           local LOCALE = node.get_attribute(item,
5626             Babel.attr_locale)
5627           local props = Babel.locale_props[LOCALE] or {}
5628           local class = Babel.cjk_class[item.char].c
5629           if props.cjk_quotes and props.cjk_quotes[item.char] then
5630             class = props.cjk_quotes[item.char]
5631           end
5632           if class == 'cp' then class = 'cl' % )) as CL
5633           elseif class == 'id' then class = 'I'
5634           elseif class == 'cj' then class = 'I' % loose
5635           end
5636           local br = 0
5637           if class and last_class and Babel.cjk_breaks[last_class][class] then
5638             br = Babel.cjk_breaks[last_class][class]
5639           end
5640           if br == 1 and props.linebreak == 'c' and
5641             lang ~= \the\l@nohyphenation\space and
5642             last_lang ~= \the\l@nohyphenation then
5643             local intrapenalty = props.intrapenalty
5644             if intrapenalty ~= 0 then
5645               local n = node.new(14, 0)      % penalty
5646               n.penalty = intrapenalty
5647               node.insert_before(head, item, n)
5648             end
5649             local intraspacespace = props.intraspacespace
5650             local n = node.new(12, 13)      % (glue, spaceskip)
5651             node.setglue(n, intraspacespace.b * quad,
5652               intraspacespace.p * quad,
5653               intraspacespace.m * quad)
5654             node.insert_before(head, item, n)
5655           end
5656           if font.getfont(item.font) then
5657             quad = font.getfont(item.font).size
5658           end
5659           last_class = class

```

```

5660         last_lang = lang
5661     else % if penalty, glue or anything else
5662         last_class = nil
5663     end
5664 end
5665 lang.hyphenate(head)
5666 end
5667 }%
5668 \bbl@lua hyphenate}
5669 \gdef\bbl@lua hyphenate{%
5670 \let\bbl@lua hyphenate\relax
5671 \directlua{
5672     luatexbase.add_to_callback('hyphenate',
5673     function (head, tail)
5674         if Babel.linebreaking.before then
5675             for k, func in ipairs(Babel.linebreaking.before) do
5676                 func(head)
5677             end
5678         end
5679         lang.hyphenate(head)
5680         if Babel.cjk_enabled then
5681             Babel.cjk_linebreak(head)
5682         end
5683         if Babel.linebreaking.after then
5684             for k, func in ipairs(Babel.linebreaking.after) do
5685                 func(head)
5686             end
5687         end
5688         if Babel.set_hboxed then
5689             Babel.set_hboxed(head)
5690         end
5691         if Babel.sea_enabled then
5692             Babel.sea_disc_to_space(head)
5693         end
5694     end,
5695     'Babel.hyphenate')
5696 }%
5697 \endgroup
5698 \def\bbl@provide@intraspace{%
5699     \bbl@ifunset\bbl@intsp\languagename\languagename\empty\else
5700     {\expandafter\ifx\csname bbl@intsp\languagename\endcsname\empty\else
5701     \bbl@xin{/c}{\bbl@cl{lnbrk}}}%
5702     \ifin@ % cjk
5703     \bbl@cjk intraspace
5704     \directlua{
5705         Babel.locale_props = Babel.locale_props or {}
5706         Babel.locale_props[\the\localeid].linebreak = 'c'
5707     }%
5708     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\languagename}%
5709     \ifx\bbl@KVP@intrapenalty\@nnil
5710         \bbl@intrapenalty0\languagename
5711     \fi
5712 \else % sea
5713     \bbl@seaintraspace
5714     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\languagename}%
5715     \directlua{
5716         Babel.sea_ranges = Babel.sea_ranges or {}
5717         Babel.set_chranges('\bbl@cl{sbcp}',
5718         '\bbl@cl{chrng}')
5719     }%
5720     \ifx\bbl@KVP@intrapenalty\@nnil
5721         \bbl@intrapenalty0\languagename
5722     \fi

```

```

5723     \fi
5724     \fi
5725     \ifx\bbl@KVP@intrapenalty\@nnil\else
5726     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5727     \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.

```

5728 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5729 \def\bblar@chars{%
5730   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5731   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5732   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5733 \def\bblar@elongated{%
5734   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5735   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5736   0649,064A}
5737 \begingroup
5738   \catcode\_ =11 \catcode\`:=11
5739   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5740 \endgroup
5741 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5742   \let\bbl@arabicjust\relax
5743   \newattribute\bblar@kashida
5744   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5745   \bblar@kashida=\z@
5746   \bbl@patchfont{\bbl@parsejalt}}%
5747   \directlua{
5748     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5749     Babel.arabic.elong_map[\the\localeid] = {}
5750     luatexbase.add_to_callback('post_linebreak_filter',
5751       Babel.arabic.justify, 'Babel.arabic.justify')
5752     luatexbase.add_to_callback('hpack_filter',
5753       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5754   }}%

```

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

```

5755 \def\bblar@fetchjalt#1#2#3#4{%
5756   \bbl@exp{\bbl@foreach{#1}}{%
5757     \bbl@ifunset\bblar@JE@##1}%
5758     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5759     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}}%
5760   \directlua{%
5761     local last = nil
5762     for item in node.traverse(tex.box[0].head) do
5763       if item.id == node.id'glyph' and item.char > 0x600 and
5764         not (item.char == 0x200D) then
5765         last = item
5766       end
5767     end
5768     Babel.arabic.#3['##1#4'] = last.char
5769   }}

```

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?

```

5770 \gdef\bbl@parsejalt{%
5771   \ifx\addfontfeature\undefined\else
5772     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5773     \ifin@
5774     \directlua{%
5775       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then

```

```

5776         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5777         tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5778     end
5779 }%
5780 \fi
5781 \fi}
5782 \gdef\bbl@parsejalti{%
5783 \beginingroup
5784 \let\bbl@parsejalt\relax % To avoid infinite loop
5785 \edef\bbl@tempb{\fontid\font}%
5786 \bblar@nofswarn
5787 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5788 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5789 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5790 \addfontfeature{RawFeature+=jalt}%
5791 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5792 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5793 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5794 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5795 \directlua{%
5796     for k, v in pairs(Babel.arabic.from) do
5797         if Babel.arabic.dest[k] and
5798             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5799             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5800             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5801         end
5802     end
5803 }%
5804 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5805 \beginingroup
5806 \catcode`#=11
5807 \catcode`~=11
5808 \directlua{
5809
5810 Babel.arabic = Babel.arabic or {}
5811 Babel.arabic.from = {}
5812 Babel.arabic.dest = {}
5813 Babel.arabic.justify_factor = 0.95
5814 Babel.arabic.justify_enabled = true
5815 Babel.arabic.kashida_limit = -1
5816
5817 function Babel.arabic.justify(head)
5818     if not Babel.arabic.justify_enabled then return head end
5819     for line in node.traverse_id(node.id'hlist', head) do
5820         Babel.arabic.justify_hlist(head, line)
5821     end
5822     return head
5823 end
5824
5825 function Babel.arabic.justify_hbox(head, gc, size, pack)
5826     local has_inf = false
5827     if Babel.arabic.justify_enabled and pack == 'exactly' then
5828         for n in node.traverse_id(12, head) do
5829             if n.stretch_order > 0 then has_inf = true end
5830         end
5831         if not has_inf then
5832             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5833         end
5834     end
5835     return head
5836 end

```

```

5837
5838 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5839     local d, new
5840     local k_list, k_item, pos_inline
5841     local width, width_new, full, k_curr, wt_pos, goal, shift
5842     local subst_done = false
5843     local elong_map = Babel.arabic.elong_map
5844     local cnt
5845     local last_line
5846     local GLYPH = node.id'glyph'
5847     local KASHIDA = Babel.attr_kashida
5848     local LOCALE = Babel.attr_locale
5849
5850     if line == nil then
5851         line = {}
5852         line.glue_sign = 1
5853         line.glue_order = 0
5854         line.head = head
5855         line.shift = 0
5856         line.width = size
5857     end
5858
5859     % Exclude last line. todo. But-- it discards one-word lines, too!
5860     % ? Look for glue = 12:15
5861     if (line.glue_sign == 1 and line.glue_order == 0) then
5862         elongs = {} % Stores elongated candidates of each line
5863         k_list = {} % And all letters with kashida
5864         pos_inline = 0 % Not yet used
5865
5866         for n in node.traverse_id(GLYPH, line.head) do
5867             pos_inline = pos_inline + 1 % To find where it is. Not used.
5868
5869             % Elongated glyphs
5870             if elong_map then
5871                 local locale = node.get_attribute(n, LOCALE)
5872                 if elong_map[locale] and elong_map[locale][n.font] and
5873                     elong_map[locale][n.font][n.char] then
5874                     table.insert(elongs, {node = n, locale = locale} )
5875                     node.set_attribute(n.prev, KASHIDA, 0)
5876                 end
5877             end
5878
5879             % Tatwil. First create a list of nodes marked with kashida. The
5880             % rest of nodes can be ignored. The list of used weights is build
5881             % when transforms with the key kashida= are declared.
5882             if Babel.kashida_wts then
5883                 local k_wt = node.get_attribute(n, KASHIDA)
5884                 if k_wt > 0 then % todo. parameter for multi inserts
5885                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5886                 end
5887             end
5888
5889             end % of node.traverse_id
5890
5891             if #elongs == 0 and #k_list == 0 then goto next_line end
5892             full = line.width
5893             shift = line.shift
5894             goal = full * Babel.arabic.justify_factor % A bit crude
5895             width = node.dimensions(line.head) % The 'natural' width
5896
5897             % == Elongated ==
5898             % Original idea taken from 'chickenize'
5899             while (#elongs > 0 and width < goal) do

```

```

5900     subst_done = true
5901     local x = #elongs
5902     local curr = elongs[x].node
5903     local oldchar = curr.char
5904     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5905     width = node.dimensions(line.head) % Check if the line is too wide
5906     % Substitute back if the line would be too wide and break:
5907     if width > goal then
5908         curr.char = oldchar
5909         break
5910     end
5911     % If continue, pop the just substituted node from the list:
5912     table.remove(elongs, x)
5913 end
5914
5915 % == Tatwil ==
5916 % Traverse the kashida node list so many times as required, until
5917 % the line is filled. The first pass adds a tatweel after each
5918 % node with kashida in the line, the second pass adds another one,
5919 % and so on. In each pass, add first the kashida with the highest
5920 % weight, then with lower weight and so on.
5921 if #k_list == 0 then goto next_line end
5922
5923 width = node.dimensions(line.head) % The 'natural' width
5924 k_curr = #k_list % Traverse backwards, from the end
5925 wt_pos = 1
5926
5927 while width < goal do
5928     subst_done = true
5929     k_item = k_list[k_curr].node
5930     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5931         d = node.copy(k_item)
5932         d.char = 0x0640
5933         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5934         d.xoffset = 0
5935         line.head, new = node.insert_after(line.head, k_item, d)
5936         width_new = node.dimensions(line.head)
5937         if width > goal or width == width_new then
5938             node.remove(line.head, new) % Better compute before
5939             break
5940         end
5941         if Babel.fix_diacr then
5942             Babel.fix_diacr(k_item.next)
5943         end
5944         width = width_new
5945     end
5946     if k_curr == 1 then
5947         k_curr = #k_list
5948         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5949     else
5950         k_curr = k_curr - 1
5951     end
5952 end
5953
5954 % Limit the number of tatweel by removing them. Not very efficient,
5955 % but it does the job in a quite predictable way.
5956 if Babel.arabic.kashida_limit > -1 then
5957     cnt = 0
5958     for n in node.traverse_id(GLYPH, line.head) do
5959         if n.char == 0x0640 then
5960             cnt = cnt + 1
5961             if cnt > Babel.arabic.kashida_limit then
5962                 node.remove(line.head, n)

```

```

5963         end
5964     else
5965         cnt = 0
5966     end
5967 end
5968 end
5969
5970 ::next_line::
5971
5972 % Must take into account marks and ins, see luatex manual.
5973 % Have to be executed only if there are changes. Investigate
5974 % what's going on exactly.
5975 if subst_done and not gc then
5976     d = node.hpack(line.head, full, 'exactly')
5977     d.shift = shift
5978     node.insert_before(head, line, d)
5979     node.remove(head, line)
5980 end
5981 end % if process line
5982 end
5983 }
5984 \endgroup
5985 \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`.

```

5986 \def\bbl@scr@node@list{%
5987   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
5988   ,Greek,Latin,Old Church Slavonic Cyrillic,}
5989 \ifnum\bbl@bidimode=102 % bidi-r
5990   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
5991 \fi
5992 \def\bbl@set@renderer{%
5993   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
5994   \ifin@
5995     \let\bbl@unset@renderer\relax
5996   \else
5997     \bbl@exp{%
5998       \def\\bbl@unset@renderer{%
5999         \def<g__fontspec_default_fontopts_clist>{%
6000           \[g__fontspec_default_fontopts_clist]}}%
6001       \def<g__fontspec_default_fontopts_clist>{%
6002         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
6003     \fi}
6004 <@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.

```

6005 % TODO - to a lua file

```

```

6006 \directlua{% DL6
6007 Babel.script_blocks = {
6008   ['dflt'] = {},
6009   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6010               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6011   ['Armn'] = {{0x0530, 0x058F}},
6012   ['Beng'] = {{0x0980, 0x09FF}},
6013   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6014   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6015   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6016              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6017   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6018   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6019              {0xAB00, 0xAB2F}},
6020   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6021   % Don't follow strictly Unicode, which places some Coptic letters in
6022   % the 'Greek and Coptic' block
6023   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6024   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6025              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6026              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6027              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6028              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6029              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6030   ['Hebr'] = {{0x0590, 0x05FF},
6031              {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6032   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6033              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6034   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6035   ['Knda'] = {{0x0C80, 0x0CFF}},
6036   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6037              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6038              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6039   ['Lao'] = {{0x0E80, 0x0EFF}},
6040   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6041              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6042              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6043   ['Mahj'] = {{0x11150, 0x1117F}},
6044   ['Mlym'] = {{0x0D00, 0x0D7F}},
6045   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6046   ['Orya'] = {{0x0B00, 0x0B7F}},
6047   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6048   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6049   ['Taml'] = {{0x0B80, 0x0BFF}},
6050   ['Telu'] = {{0x0C00, 0x0C7F}},
6051   ['Tfng'] = {{0x2D30, 0x2D7F}},
6052   ['Thai'] = {{0x0E00, 0x0E7F}},
6053   ['Tibt'] = {{0x0F00, 0x0FFF}},
6054   ['Vaii'] = {{0xA500, 0xA63F}},
6055   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6056 }
6057
6058 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6059 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6060 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6061
6062 function Babel.locale_map(head)
6063   if not Babel.locale_mapped then return head end
6064
6065   local LOCALE = Babel.attr_locale
6066   local GLYPH = node.id('glyph')
6067   local inmath = false
6068   local toloc_save

```



```

6069 for item in node.traverse(head) do
6070   local toloc
6071   if not inmath and item.id == GLYPH then
6072     % Optimization: build a table with the chars found
6073     if Babel.chr_to_loc[item.char] then
6074       toloc = Babel.chr_to_loc[item.char]
6075     else
6076       for lc, maps in pairs(Babel.loc_to_scr) do
6077         for _, rg in pairs(maps) do
6078           if item.char >= rg[1] and item.char <= rg[2] then
6079             Babel.chr_to_loc[item.char] = lc
6080             toloc = lc
6081             break
6082           end
6083         end
6084       end
6085       % Treat composite chars in a different fashion, because they
6086       % 'inherit' the previous locale.
6087       if (item.char >= 0x0300 and item.char <= 0x036F) or
6088          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6089          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6090         Babel.chr_to_loc[item.char] = -2000
6091         toloc = -2000
6092       end
6093       if not toloc then
6094         Babel.chr_to_loc[item.char] = -1000
6095       end
6096     end
6097     if toloc == -2000 then
6098       toloc = toloc_save
6099     elseif toloc == -1000 then
6100       toloc = nil
6101     end
6102     if toloc and Babel.locale_props[toloc] and
6103        Babel.locale_props[toloc].letters and
6104        tex.getcatcode(item.char) \string~= 11 then
6105       toloc = nil
6106     end
6107     if toloc and Babel.locale_props[toloc].script
6108        and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6109        and Babel.locale_props[toloc].script ==
6110        Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6111       toloc = nil
6112     end
6113     if toloc then
6114       if Babel.locale_props[toloc].lg then
6115         item.lang = Babel.locale_props[toloc].lg
6116         node.set_attribute(item, LOCALE, toloc)
6117       end
6118       if Babel.locale_props[toloc]['/'..item.font] then
6119         item.font = Babel.locale_props[toloc]['/'..item.font]
6120       end
6121     end
6122     toloc_save = toloc
6123   elseif not inmath and item.id == 7 then % Apply recursively
6124     item.replace = item.replace and Babel.locale_map(item.replace)
6125     item.pre      = item.pre and Babel.locale_map(item.pre)
6126     item.post     = item.post and Babel.locale_map(item.post)
6127   elseif item.id == node.id'math' then
6128     inmath = (item.subtype == 0)
6129   end
6130 end
6131 return head

```

```
6132 end
6133 }
```

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

```
6134 \newcommand\babelcharproperty[1]{%
6135   \count@=#1\relax
6136   \ifvmode
6137     \expandafter\babel@chprop
6138   \else
6139     \babel@error{charproperty-only-vertical}{\count@}{\count@}%
6140   \fi}
6141 \newcommand\babel@chprop[3][\the\count@]{%
6142   \@tempcnta=#1\relax
6143   \babel@ifunset{babel@chprop@#2}% {unknown-char-property}
6144   {\babel@error{unknown-char-property}{\count@}{\count@}%
6145    \count@}%
6146   \loop
6147     \babel@cs{chprop@#2}{\count@}%
6148   \ifnum\count@<\@tempcnta
6149     \advance\count@\@ne
6150   \repeat}
6151 \def\babel@chprop@direction#1{%
6152   \directlua{
6153     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6154     Babel.characters[\the\count@]['d'] = '#1'
6155   }}
6156 \let\babel@chprop@bc\babel@chprop@direction
6157 \def\babel@chprop@mirror#1{%
6158   \directlua{
6159     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6160     Babel.characters[\the\count@]['m'] = '\number#1'
6161   }}
6162 \let\babel@chprop@bmg\babel@chprop@mirror
6163 \def\babel@chprop@linebreak#1{%
6164   \directlua{
6165     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6166     Babel.cjk_characters[\the\count@]['c'] = '#1'
6167   }}
6168 \let\babel@chprop@lb\babel@chprop@linebreak
6169 \def\babel@chprop@locale#1{%
6170   \directlua{
6171     Babel.chr_to_loc = Babel.chr_to_loc or {}
6172     Babel.chr_to_loc[\the\count@] =
6173       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
6174   }}
```

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.

```
6175 \directlua{% DL7
6176   Babel.nohyphenation = \the\l@nohyphenation
6177 }
```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the `{n}` syntax. For example, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect 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).

```
6178 \begingroup
6179 \catcode\~ = 12
```

```

6180 \catcode\%=12
6181 \catcode\&=14
6182 \catcode\|=12
6183 \gdef\babelprehyphenation{%&
6184 \@ifnextchar{\bbl@settransform{0}}{\bbl@settransform{0}}{}}
6185 \gdef\babelposthyphenation{%&
6186 \@ifnextchar{\bbl@settransform{1}}{\bbl@settransform{1}}{}}
6187 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6188 \ifcase#1
6189 \bbl@activateprehyphen
6190 \or
6191 \bbl@activateposthyphen
6192 \fi
6193 \begingroup
6194 \def\babeltempa{\bbl@add@list\babeltempb}%&
6195 \let\babeltempb\empty
6196 \def\bbl@tempa{#5}%&
6197 \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6198 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6199 \bbl@ifsamestring{##1}{remove}%&
6200 {\bbl@add@list\babeltempb{nil}}}%&
6201 {\directlua{
6202 local rep = [=([##1])=]
6203 local three_args = '%s*=%s*([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)'
6204 & Numeric passes directly: kern, penalty...
6205 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6206 rep = rep:gsub('^%s*(insert)%s*', 'insert = true,')
6207 rep = rep:gsub('^%s*(after)%s*', 'after = true,')
6208 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6209 rep = rep:gsub('node%s*=%s*([a+])%s*([a+])', Babel.capture_node)
6210 rep = rep:gsub(' (norule)' .. three_args,
6211 'norule = {' .. '%2, %3, %4' .. '})')
6212 if #1 == 0 or #1 == 2 then
6213 rep = rep:gsub(' (space)' .. three_args,
6214 'space = {' .. '%2, %3, %4' .. '})')
6215 rep = rep:gsub(' (spacefactor)' .. three_args,
6216 'spacefactor = {' .. '%2, %3, %4' .. '})')
6217 rep = rep:gsub(' (kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6218 & Transform values
6219 rep, n = rep:gsub(' {([%a%-%.]+)|([%a%_%.]+)}',
6220 function(v,d)
6221 return string.format (
6222 '\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6223 v,
6224 load( 'return Babel.locale_props'..
6225 '\the\csname bbl@id@@#3\endcsname].' .. d)() )
6226 end )
6227 rep, n = rep:gsub(' {([%a%-%.]+)|([%-d%.]+)}',
6228 '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6229 end
6230 if #1 == 1 then
6231 rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6232 rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6233 rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6234 end
6235 tex.print([[\string\babeltempa{[]] .. rep .. [[]]])
6236 }}}%&
6237 \bbl@foreach\babeltempb{%&
6238 \bbl@forkv{##1}}{%&
6239 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&
6240 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}%&
6241 \ifin@{else
6242 \bbl@error{bad-transform-option}{###1}}{}}{%&

```

```

6243     \fi}}&%
6244 \let\bbl@kv@attribute\relax
6245 \let\bbl@kv@label\relax
6246 \let\bbl@kv@fonts\empty
6247 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6248 \ifx\bbl@kv@fonts\empty\else\bbl@settransformfont\fi
6249 \ifx\bbl@kv@attribute\relax
6250   \ifx\bbl@kv@label\relax\else
6251     \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6252     \bbl@replace\bbl@kv@fonts{ }{,}&%
6253     \edef\bbl@kv@attribute{\bbl@ATR@{\bbl@kv@label @#3@\bbl@kv@fonts}}&%
6254     \count@=\z@
6255     \def\bbl@elt##1##2##3{&%
6256       \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6257       {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6258         {\count@\ne}&%
6259         {\bbl@error{font-conflict-transforms}{}}}&%
6260       {}}&%
6261     \bbl@transformfont@list
6262     \ifnum\count@=\z@
6263       \bbl@exp{\global\bbl@add\bbl@transformfont@list
6264         {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6265       \fi
6266       \bbl@ifunset{\bbl@kv@attribute}&%
6267       {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6268       {}}&%
6269       \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6270     \fi
6271 \else
6272   \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6273 \fi
6274 \directlua{
6275   local lbkr = Babel.linebreaking.replacements[#1]
6276   local u = unicode.utf8
6277   local id, attr, label
6278   if #1 == 0 then
6279     id = \the\csname bbl@id@@#3\endcsname\space
6280   else
6281     id = \the\csname l@#3\endcsname\space
6282   end
6283   \ifx\bbl@kv@attribute\relax
6284     attr = -1
6285   \else
6286     attr = luatexbase.registernumber'\bbl@kv@attribute'
6287   \fi
6288   \ifx\bbl@kv@label\relax\else &% Same refs:
6289     label = [==[\bbl@kv@label]==]
6290   \fi
6291   &% Convert pattern:
6292   local patt = string.gsub([==[#4]==], '%s', '')
6293   if #1 == 0 then
6294     patt = string.gsub(patt, '|', ' ')
6295   end
6296   if not u.find(patt, '()', nil, true) then
6297     patt = '()' .. patt .. '()'
6298   end
6299   if #1 == 1 then
6300     patt = string.gsub(patt, '%(%)^', '^()')
6301     patt = string.gsub(patt, '%$(%)', '()$')
6302   end
6303   patt = u.gsub(patt, '{(.)}',
6304     function (n)
6305       return '%' .. (tonumber(n) and (tonumber(n)+1) or n)

```

```

6306         end)
6307     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6308         function (n)
6309             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6310         end)
6311     lbkr[id] = lbkr[id] or {}
6312     table.insert(lbkr[id],
6313         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6314 }&%
6315 \endgroup}
6316 \endgroup
6317 \let\bbl@transfont@list\@empty
6318 \def\bbl@settransfont{%
6319     \global\let\bbl@settransfont\relax % Execute only once
6320     \gdef\bbl@transfont{%
6321         \def\bbl@elt####1####2####3{%
6322             \bbl@ifblank{####3}%
6323             {\count\@tw@}% Do nothing if no fonts
6324             {\count\@z@
6325                 \bbl@vforeach{####3}{%
6326                     \def\bbl@tempd{#####1}%
6327                     \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6328                     \ifx\bbl@tempd\bbl@tempe
6329                         \count\@ne
6330                     \else\ifx\bbl@tempd\bbl@transfam
6331                         \count\@ne
6332                     \fi\fi}%
6333                 \ifcase\count@
6334                     \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6335                 \or
6336                     \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6337                 \fi}}%
6338                 \bbl@transfont@list}%
6339     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6340     \gdef\bbl@transfam{-unknown-}%
6341     \bbl@foreach\bbl@font@fams{%
6342         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6343         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6344         {\xdef\bbl@transfam{##1}}%
6345     }}}}
6346 \DeclareRobustCommand\enablelocaletransform[1]{%
6347     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6348     {\bbl@error{transform-not-available}{#1}}}%
6349     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6350 \DeclareRobustCommand\disablelocaletransform[1]{%
6351     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6352     {\bbl@error{transform-not-available-b}{#1}}}%
6353     {\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`.

```

6354 \def\bbl@activateposthyphen{%
6355     \let\bbl@activateposthyphen\relax
6356     \ifx\bbl@attr@hboxed\undefined
6357         \newattribute\bbl@attr@hboxed
6358     \fi
6359     \directlua{
6360         require('babel-transforms.lua')
6361         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6362     }}
6363 \def\bbl@activateprehyphen{%
6364     \let\bbl@activateprehyphen\relax
6365     \ifx\bbl@attr@hboxed\undefined

```

```

6366 \newattribute\bbl@attr@hboxed
6367 \fi
6368 \directlua{
6369   require('babel-transforms.lua')
6370   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6371 }
6372 \newcommand\SetTransformValue[3]{%
6373 \directlua{
6374   Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6375 }}

```

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.

```

6376 \newcommand\ShowBabelTransforms[1]{%
6377 \bbl@activateprehyphen
6378 \bbl@activateposthyphen
6379 \beginngroup
6380 \directlua{ Babel.show_transforms = true }%
6381 \setbox\z@\vbox{#1}%
6382 \directlua{ Babel.show_transforms = false }%
6383 \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.

```

6384 \newcommand\localeprehyphenation[1]{%
6385 \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.

```

6386 \def\bbl@activate@preotf{%
6387 \let\bbl@activate@preotf\relax % only once
6388 \directlua{
6389   function Babel.pre_otfload_v(head)
6390     if Babel.numbers and Babel.digits_mapped then
6391       head = Babel.numbers(head)
6392     end
6393     if Babel.bidi_enabled then
6394       head = Babel.bidi(head, false, dir)
6395     end
6396     return head
6397   end
6398   %
6399   function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6400     if Babel.numbers and Babel.digits_mapped then
6401       head = Babel.numbers(head)
6402     end
6403     if Babel.bidi_enabled then
6404       head = Babel.bidi(head, false, dir)
6405     end
6406     return head
6407   end
6408   %
6409   luatexbase.add_to_callback('pre_linebreak_filter',
6410     Babel.pre_otfload_v,
6411     'Babel.pre_otfload_v',
6412     luatexbase.priority_in_callback('pre_linebreak_filter',

```

```

6413     'luaotfload.node_processor') or nil)
6414 %
6415     luatexbase.add_to_callback('hpack_filter',
6416         Babel.pre_otfload_h,
6417         'Babel.pre_otfload_h',
6418         luatexbase.priority_in_callback('hpack_filter',
6419             'luaotfload.node_processor') or nil)
6420 }}

```

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`.

```

6421 \breakafterdirmode=1
6422 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6423   \let\bbl@beforeforeign\leavevmode
6424   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6425   \RequirePackage{luatexbase}
6426   \bbl@activate@preotf
6427   \directlua{
6428     require('babel-data-bidi.lua')
6429     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6430       require('babel-bidi-basic.lua')
6431     \or
6432       require('babel-bidi-basic-r.lua')
6433     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6434     table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6435     table.insert(Babel.ranges, {0x10000, 0x10FFFFD, 'on'})
6436   \fi}
6437   \newattribute\bbl@attr@dir
6438   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6439   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6440 \fi
6441 \chardef\bbl@thetextdir\z@
6442 \chardef\bbl@thepardir\z@
6443 \def\bbl@getluadir#1{%
6444   \directlua{
6445     if tex.#ldir == 'TLT' then
6446       tex.sprint('0')
6447     elseif tex.#ldir == 'TRT' then
6448       tex.sprint('1')
6449     else
6450       tex.sprint('0')
6451     end}}
6452 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6453   \ifcase#3\relax
6454     \ifcase\bbl@getluadir{#1}\relax\else
6455       #2 TLT\relax
6456     \fi
6457   \else
6458     \ifcase\bbl@getluadir{#1}\relax
6459       #2 TRT\relax
6460     \fi
6461   \fi}
6462 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6463 \def\bbl@thedir{0}
6464 \def\bbl@textdir#1{%
6465   \bbl@setluadir{text}\textdir{#1}%
6466   \chardef\bbl@thetextdir#1\relax
6467   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6468   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6469 \def\bbl@pardir#1{% Used twice
6470   \bbl@setluadir{par}\pardir{#1}%

```

```

6471 \chardef\bbl@thepardir#1\relax}
6472 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6473 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6474 \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.

6475 \ifnum\bbl@bidimode>\z@ % Any bidi=
6476 \def\bbl@insidemath{0}%
6477 \def\bbl@everymath{\def\bbl@insidemath{1}}
6478 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6479 \frozen@everymath\expandafter{%
6480 \expandafter\bbl@everymath\the\frozen@everymath}
6481 \frozen@everydisplay\expandafter{%
6482 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6483 \AtBeginDocument{
6484 \directlua{
6485 function Babel.math_box_dir(head)
6486 if not (token.get_macro('bbl@insidemath') == '0') then
6487 if Babel.hlist_has_bidi(head) then
6488 local d = node.new(node.id'dir')
6489 d.dir = '+TRT'
6490 node.insert_before(head, node.has_glyph(head), d)
6491 local inmath = false
6492 for item in node.traverse(head) do
6493 if item.id == 11 then
6494 inmath = (item.subtype == 0)
6495 elseif not inmath then
6496 node.set_attribute(item,
6497 Babel.attr_dir, token.get_macro('bbl@thedir'))
6498 end
6499 end
6500 end
6501 end
6502 return head
6503 end
6504 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6505 "Babel.math_box_dir", 0)
6506 if Babel.unset_atdir then
6507 luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6508 "Babel.unset_atdir")
6509 luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6510 "Babel.unset_atdir")
6511 end
6512 }}%
6513 \fi

Experimental. Tentative name.

6514 \DeclareRobustCommand\localebox[1]{%
6515 {\def\bbl@insidemath{0}%
6516 \mbox{\foreignlanguage{\language}\language\{#1}}}}

```

10.12.Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `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.

```

6517 \bbl@trace{Redefinitions for bidi layout}
6518 %
6519 <<{*More package options}>> ≡
6520 \chardef\bbl@eqnpos\z@
6521 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6522 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6523 <</More package options>>
6524 %
6525 \ifnum\bbl@bidimode>\z@ % Any bidi=
6526 \matheqdirmode\@ne % A luatex primitive
6527 \let\bbl@eqnodir\relax
6528 \def\bbl@eqdel{()}
6529 \def\bbl@eqnum{%
6530   {\normalfont\normalcolor
6531     \expandafter\@firstoftwo\bbl@eqdel
6532     \theequation
6533     \expandafter\@secondoftwo\bbl@eqdel}}
6534 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6535 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6536 \def\bbl@eqno@flip#1{%
6537   \ifdim\predisplaysize=-\maxdimen
6538     \eqno
6539     \hb@xt@.01pt{%
6540       \hb@xt@\displaywidth{\hss{#1\glet\bbl@upset\@currentlabel}}\hss}%
6541   \else
6542     \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6543   \fi
6544   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}}
6545 \def\bbl@leqno@flip#1{%
6546   \ifdim\predisplaysize=-\maxdimen
6547     \leqno
6548     \hb@xt@.01pt{%
6549       \hss\hb@xt@\displaywidth{\#1\glet\bbl@upset\@currentlabel}\hss}%
6550   \else
6551     \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6552   \fi
6553   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}}
6554 \AtBeginDocument{%
6555   \ifx\bbl@noamsmath\relax\else
6556   \ifx\maketag@@@\undefined % Normal equation, eqnarray
6557     \AddToHook{env/equation/begin}{%
6558       \ifnum\bbl@thetextdir>\z@
6559         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6560         \let\@eqnnum\bbl@eqnum
6561         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6562         \chardef\bbl@thetextdir\z@
6563         \bbl@add\normalfont{\bbl@eqnodir}%
6564         \ifcase\bbl@eqnpos
6565           \let\bbl@puteqno\bbl@eqno@flip
6566         \or
6567           \let\bbl@puteqno\bbl@leqno@flip
6568         \fi
6569       \fi}%

```

```

6570 \ifnum\bbledqnpos=\tw@ \else
6571 \def\endequation{\bbledputeqno{\@eqnnum}$$\@ignoretrue}%
6572 \fi
6573 \AddToHook{env/eqnarray/begin}{%
6574 \ifnum\bbledthetextdir>\z@
6575 \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6576 \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6577 \chardef\bbledthetextdir\z@
6578 \bbledadd\normalfont{\bbledeqnodir}%
6579 \ifnum\bbledqnpos=\@ne
6580 \def\@eqnnum{%
6581 \setbox\z@\hbox{\bbledeqnum}%
6582 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6583 \else
6584 \let\@eqnnum\bbledeqnum
6585 \fi
6586 \fi}
6587 % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6588 \expandafter\bbledsreplace\csname \endcsname{$$}{eqno\kern.001pt$$}%
6589 \else % amstex
6590 \bbledexp{% Hack to hide maybe undefined conditionals:
6591 \chardef\bbledqnpos=0%
6592 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6593 \ifnum\bbledqnpos=\@ne
6594 \let\bbledams@lap\hbox
6595 \else
6596 \let\bbledams@lap\llap
6597 \fi
6598 \ExplSyntaxOn % Required by \bbledsreplace with \intertext@
6599 \bbledsreplace\intertext@\{normalbaselines}%
6600 {\normalbaselines
6601 \ifx\bbledeqnodir\relax\else\bbledpdir\@ne\bbledeqnodir\fi}%
6602 \ExplSyntaxOff
6603 \def\bbledams@tagbox#1#2{\#1{\bbledeqnodir#2}}% #1=hbox|@lap|flip
6604 \ifx\bbledams@lap\hbox % leqno
6605 \def\bbledams@flip#1{%
6606 \hbox to 0.01pt{\hss\hbox to\displaywidth{\#1\hss}}}%
6607 \else % eqno
6608 \def\bbledams@flip#1{%
6609 \hbox to 0.01pt{\hbox to\displaywidth{\hss\#1\hss}}}%
6610 \fi
6611 \def\bbledams@preset#1{%
6612 \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6613 \ifnum\bbledthetextdir>\z@
6614 \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6615 \bbledsreplace\textdef@\{\hbox}\{\bbledams@tagbox\hbox}%
6616 \bbledsreplace\maketag@@@\{\hbox}\{\bbledams@tagbox#1}%
6617 \fi}%
6618 \ifnum\bbledqnpos=\tw@ \else
6619 \def\bbledams@equation{%
6620 \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6621 \ifnum\bbledthetextdir>\z@
6622 \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6623 \chardef\bbledthetextdir\z@
6624 \bbledadd\normalfont{\bbledeqnodir}%
6625 \ifcase\bbledqnpos
6626 \def\veqno##1##2{\bbledeqno@flip{##1##2}}%
6627 \or
6628 \def\veqno##1##2{\bbledleqno@flip{##1##2}}%
6629 \fi
6630 \fi}%
6631 \AddToHook{env/equation/begin}{\bbledams@equation}%
6632 \AddToHook{env/equation*/begin}{\bbledams@equation}%

```

```

6633 \fi
6634 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6635 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6636 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6637 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6638 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6639 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6640 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6641 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6642 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6643 % Hackish, for proper alignment. Don't ask me why it works!:
6644 \bbl@exp{% Avoid a 'visible' conditional
6645   \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{}\<fi>}%
6646   \\\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{}\<fi>}}%
6647 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6648 \AddToHook{env/split/before}{%
6649   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6650   \ifnum\bbl@thetextdir>\z@
6651     \bbl@ifsamestring\@currentenv{equation}%
6652     {\ifx\bbl@ams@lap\hbox % leqno
6653       \def\bbl@ams@flip#1{%
6654         \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6655       \else
6656       \def\bbl@ams@flip#1{%
6657         \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6658       \fi}%
6659     }%
6660   \fi}%
6661 \fi\fi}
6662 \fi
6663 \def\bbl@provide@extra#1{%
6664   % == onchar ==
6665   \ifx\bbl@KVP@onchar\@nnil\else
6666     \bbl@luahyphenate
6667     \bbl@exp{%
6668       \\\AddToHook{env/document/before}{\select@language{#1}}}%
6669     \directlua{
6670       if Babel.locale_mapped == nil then
6671         Babel.locale_mapped = true
6672         Babel.linebreaking.add_before(Babel.locale_map, 1)
6673         Babel.loc_to_scr = {}
6674         Babel.chr_to_loc = Babel.chr_to_loc or {}
6675       end
6676       Babel.locale_props[\the\localeid].letters = false
6677     }%
6678     \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6679     \ifin@
6680       \directlua{
6681         Babel.locale_props[\the\localeid].letters = true
6682       }%
6683     \fi
6684     \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6685     \ifin@
6686       \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6687         \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6688       \fi
6689       \bbl@exp{\\bbl@add\\bbl@starthyphens
6690         {\\bbl@patterns@lua{\languagename}}}%
6691       %^A add error/warning if no script
6692       \directlua{
6693         if Babel.script_blocks['\bbl@cl{sbc}'] then
6694           Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6695           Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space

```

```

6696         end
6697     }%
6698 \fi
6699 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6700 \ifin@
6701     \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{}%
6702     \bbl@ifunset{bbl@wdir@\languageName}{\bbl@provide@dirs{\languageName}}{}%
6703     \directlua{
6704         if Babel.script_blocks['\bbl@cl{sbc}'] then
6705             Babel.loc_to_scr[\the\localeid] =
6706                 Babel.script_blocks['\bbl@cl{sbc}']
6707         end}%
6708     \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
6709     \AtBeginDocument{%
6710         \bbl@patchfont{\bbl@mapselect}}%
6711         {\selectfont}}%
6712     \def\bbl@mapselect{%
6713         \let\bbl@mapselect\relax
6714         \edef\bbl@prefontid{\fontid\font}}%
6715     \def\bbl@mapdir##1{%
6716         \begingroup
6717             \setbox\z@\hbox{% Force text mode
6718                 \def\languageName{##1}%
6719                 \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6720                 \bbl@switchfont
6721                 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6722                     \directlua{
6723                         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6724                             ['/\bbl@prefontid'] = \fontid\font\space}%
6725                     \fi}%
6726             \endgroup}%
6727     \fi
6728     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languageName}}}%
6729 \fi
6730 % TODO - catch non-valid values
6731 \fi
6732 % == mapfont ==
6733 % For bidi texts, to switch the font based on direction. Old.
6734 \ifx\bbl@KVP@mapfont\@nnil\else
6735     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
6736     {\bbl@error{unknown-mapfont}{}}{}%
6737     \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{}%
6738     \bbl@ifunset{bbl@wdir@\languageName}{\bbl@provide@dirs{\languageName}}{}%
6739     \ifx\bbl@mapselect\undefined % TODO. See onchar.
6740     \AtBeginDocument{%
6741         \bbl@patchfont{\bbl@mapselect}}%
6742         {\selectfont}}%
6743     \def\bbl@mapselect{%
6744         \let\bbl@mapselect\relax
6745         \edef\bbl@prefontid{\fontid\font}}%
6746     \def\bbl@mapdir##1{%
6747         {\def\languageName{##1}%
6748         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6749         \bbl@switchfont
6750         \directlua{Babel.fontmap
6751             [\the\csname bbl@wdir@##1\endcsname]%
6752             [\bbl@prefontid]=\fontid\font}}}%
6753     \fi
6754     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languageName}}}%
6755 \fi
6756 % == Line breaking: CJK quotes ==
6757 \ifcase\bbl@engine\or
6758     \bbl@xin@{/c}{\bbl@cl{\lnbrk}}%

```

```

6759 \ifin@
6760 \bbl@ifunset{bbl@quote@\language\name}{}%
6761 {\directlua{
6762   Babel.locale_props[\the\localeid].CJK_quotes = {}
6763   local cs = 'op'
6764   for c in string.utfvalues(
6765     [[\csname bbl@quote@\language\name\endcsname]]) do
6766     if Babel.CJK_characters[c].c == 'qu' then
6767       Babel.locale_props[\the\localeid].CJK_quotes[c] = cs
6768     end
6769     cs = (cs == 'op') and 'cl' or 'op'
6770   end
6771 }}%
6772 \fi
6773 \fi
6774 % == Counters: mapdigits ==
6775 % Native digits
6776 \ifx\bbl@KVP@mapdigits\@nnil\else
6777 \bbl@ifunset{bbl@dgnat@\language\name}{}%
6778 {\RequirePackage{luatexbase}%
6779 \bbl@activate@preotf
6780 \directlua{
6781   Babel.digits_mapped = true
6782   Babel.digits = Babel.digits or {}
6783   Babel.digits[\the\localeid] =
6784     table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6785   if not Babel.numbers then
6786     function Babel.numbers(head)
6787       local LOCALE = Babel.attr_locale
6788       local GLYPH = node.id'glyph'
6789       local inmath = false
6790       for item in node.traverse(head) do
6791         if not inmath and item.id == GLYPH then
6792           local temp = node.get_attribute(item, LOCALE)
6793           if Babel.digits[temp] then
6794             local chr = item.char
6795             if chr > 47 and chr < 58 then
6796               item.char = Babel.digits[temp][chr-47]
6797             end
6798           end
6799         elseif item.id == node.id'math' then
6800           inmath = (item.subtype == 0)
6801         end
6802       end
6803       return head
6804     end
6805   end
6806 }}%
6807 \fi
6808 % == transforms ==
6809 \ifx\bbl@KVP@transforms\@nnil\else
6810 \def\bbl@elt##1##2##3{%
6811   \in@{${transforms.}}{##1}%
6812   \ifin@
6813     \def\bbl@tempa{##1}%
6814     \bbl@replace\bbl@tempa{transforms.}%
6815     \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6816   \fi}%
6817 \bbl@exp{%
6818   \\bbl@ifblank{\bbl@cl{dgnat}}%
6819   {\let\\bbl@tempa\relax}%
6820   {\def\\bbl@tempa{%
6821     \\bbl@elt{transforms.prehyphenation}%

```

```

6822         {digits.native.1.0}{([0-9])}%
6823         \\bbl@elt{transforms.prehyphenation}%
6824         {digits.native.1.1}{string={\string|0123456789\string|bbl@cl{dgnat}}}}}%
6825     \ifx\bbl@tempa\relax\else
6826         \toks@{\expandafter\expandafter\expandafter{%
6827             \csname bbl@inidata@\languagename\endcsname}%
6828             \bbl@csarg\edef{inidata@\languagename}{%
6829                 \unexpanded\expandafter{\bbl@tempa}%
6830                 \the\toks@}%
6831         \fi
6832         \csname bbl@inidata@\languagename\endcsname
6833         \bbl@release@transforms\relax % \relax closes the last item.
6834     \fi}

```

Start tabular here:

```

6835 \def\localerestoredirs{%
6836     \ifcase\bbl@thetextdir
6837         \ifnum\textdirection=\z@\else\textdir TLT\fi
6838     \else
6839         \ifnum\textdirection=\@ne\else\textdir TRT\fi
6840     \fi
6841     \ifcase\bbl@thepardir
6842         \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6843     \else
6844         \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6845     \fi}
6846 \IfBabelLayout{tabular}%
6847     {\chardef\bbl@tabular@mode\tw@}% All RTL
6848     {\IfBabelLayout{notabular}%
6849         {\chardef\bbl@tabular@mode\z@}%
6850         {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6851 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6852 % Redefine: vrules mess up dirs. TODO: why?
6853 \def\@arstrut{\relax\copy\@arstrutbox}%
6854 \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6855     \let\bbl@parabefore\relax
6856     \AddToHook{para/before}{\bbl@parabefore}
6857     \AtBeginDocument{%
6858         \bbl@replace\@tabular{${}}{${}
6859             \def\bbl@insidemath{0}%
6860             \def\bbl@parabefore{\localerestoredirs}}}%
6861     \ifnum\bbl@tabular@mode=\@ne
6862         \bbl@ifunset{\@tabclassz}{}%
6863         \bbl@exp{% Hide conditionals
6864             \\bbl@sreplace\\ \@tabclassz
6865             {\<ifcase>\\ \@chnum}%
6866             {\localerestoredirs\<ifcase>\\ \@chnum}}}%
6867     \@ifpackage{colortbl}%
6868     {\bbl@sreplace\@classz
6869         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6870     {\@ifpackage{array}%
6871         {\bbl@exp{% Hide conditionals
6872             \\bbl@sreplace\\ \@classz
6873             {\<ifcase>\\ \@chnum}%
6874             {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6875             \\bbl@sreplace\\ \@classz
6876             {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6877         {}}}%
6878     \fi}%
6879 \or % 2 = All RTL - tabular
6880     \let\bbl@parabefore\relax
6881     \AddToHook{para/before}{\bbl@parabefore}%
6882     \AtBeginDocument{%

```

```

6883 \ifpackageloaded{colortbl}%
6884 {\bbl@replace\@tabular{$}{$}%
6885 \def\bbl@insidemath{0}%
6886 \def\bbl@parabefore{\localerestoredirs}}%
6887 \bbl@sreplace\@classz
6888 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6889 {}}%
6890 \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.

```

6891 \AtBeginDocument{%
6892 \ifpackageloaded{multicol}%
6893 {\toks@expandafter{\multi@column@out}%
6894 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6895 {}%
6896 \ifpackageloaded{paracol}%
6897 {\edef\pcol@output{%
6898 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6899 {}}%
6900 \fi
6901 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

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

```

6902 \ifnum\bbl@bidimode>\z@ % Any bidi=
6903 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6904 \bbl@exp{%
6905 \mathdir\the\bodydir
6906 #1% Once entered in math, set boxes to restore values
6907 \def\\bbl@insidemath{0}%
6908 \<ifmmode>%
6909 \everyvbox{%
6910 \the\everyvbox
6911 \bodydir\the\bodydir
6912 \mathdir\the\mathdir
6913 \everyhbox{\the\everyhbox}%
6914 \everyvbox{\the\everyvbox}}%
6915 \everyhbox{%
6916 \the\everyhbox
6917 \bodydir\the\bodydir
6918 \mathdir\the\mathdir
6919 \everyhbox{\the\everyhbox}%
6920 \everyvbox{\the\everyvbox}}%
6921 \<fi>}}%
6922 \def\@hangfrom#1{%
6923 \setbox\@tempboxa\hbox{#1}}%
6924 \hangindent\wd\@tempboxa
6925 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6926 \shapemode\@ne
6927 \fi
6928 \noindent\box\@tempboxa}
6929 \fi
6930 \IfBabelLayout{tabular}
6931 {\let\bbl@OL@tabular\@tabular
6932 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6933 \let\bbl@NL@tabular\@tabular
6934 \AtBeginDocument{%
6935 \ifx\bbl@NL@tabular\@tabular\else
6936 \bbl@exp{\in{\bbl@nextfake}{\@tabular}}}%
6937 \ifin@else

```

```

6938      \bbl@replace\@tabular{$}\bbl@nextfakes$}%
6939      \fi
6940      \let\bbl@NL@tabular\@tabular
6941      \fi}}
6942      {}
6943 \IfBabelLayout{lists}
6944 {\let\bbl@OL@list\list
6945  \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6946  \let\bbl@NL@list\list
6947  \def\bbl@listparshape#1#2#3{%
6948    \parshape #1 #2 #3 %
6949    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6950      \shapemode\tw@
6951    \fi}}
6952  {}
6953 \IfBabelLayout{graphics}
6954 {\let\bbl@pictresetdir\relax
6955  \def\bbl@pictsetdir#1{%
6956    \ifcase\bbl@thetextdir
6957      \let\bbl@pictresetdir\relax
6958    \else
6959      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6960        \or\textdir TLT
6961        \else\bodydir TLT \textdir TLT
6962      \fi
6963      % \text\par\dir required in pgf:
6964      \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6965    \fi}%
6966 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6967 \directlua{
6968   Babel.get_picture_dir = true
6969   Babel.picture_has_bidi = 0
6970   %
6971   function Babel.picture_dir (head)
6972     if not Babel.get_picture_dir then return head end
6973     if Babel.hlist_has_bidi(head) then
6974       Babel.picture_has_bidi = 1
6975     end
6976     return head
6977   end
6978   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6979     "Babel.picture_dir")
6980 }%
6981 \AtBeginDocument{%
6982   \def\LS@rot{%
6983     \setbox\@outputbox\vbox{%
6984       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6985   \long\def\put(#1,#2)#3{%
6986     \@killglue
6987     % Try:
6988     \ifx\bbl@pictresetdir\relax
6989       \def\bbl@tempc{0}%
6990     \else
6991       \directlua{
6992         Babel.get_picture_dir = true
6993         Babel.picture_has_bidi = 0
6994       }%
6995       \setbox\z@\hb@xt@z@{%
6996         \@defaultunitsset\@tempdimc{#1}\unitlength
6997         \kern\@tempdimc
6998         #3\hss}% TODO: #3 executed twice (below). That's bad.
6999       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7000     \fi

```



```

7001 % Do:
7002 \@defaultunitsset\@tempdimc{#2}\unitlength
7003 \raise\@tempdimc\hbxt\z@{%
7004 \@defaultunitsset\@tempdimc{#1}\unitlength
7005 \kern\@tempdimc
7006 {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
7007 \ignorespaces}%
7008 \MakeRobust\put}%
7009 \AtBeginDocument
7010 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
7011 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
7012 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
7013 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
7014 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
7015 \fi
7016 \ifx\tikzpicture\@undefined\else
7017 \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
7018 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
7019 \bbbl@sreplace\tikz{\beginpgroup}{\beginpgroup\bbbl@pictsetdir\tw@}%
7020 \bbbl@sreplace\tikzpicture{\beginpgroup}{\beginpgroup\bbbl@pictsetdir\tw@}%
7021 \fi
7022 \ifx\tcolorbox\@undefined\else
7023 \def\tcb@drawing@env@begin{%
7024 \csname tcb@before@\tcb@split@state\endcsname
7025 \bbbl@pictsetdir\tw@
7026 \begin{\kvtcb@graphenv}%
7027 \tcb@bbdraw
7028 \tcb@apply@graph@patches}%
7029 \def\tcb@drawing@env@end{%
7030 \end{\kvtcb@graphenv}%
7031 \bbbl@pictresetdir
7032 \csname tcb@after@\tcb@split@state\endcsname}%
7033 \fi
7034 }}
7035 {}

```

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.

```

7036 \IfBabelLayout{counters*}%
7037 {\bbbl@add\bbbl@opt@layout{.counters.}%
7038 \directlua{
7039 \lua{
7040 \lua{
7041 }{}
7042 \IfBabelLayout{counters*}%
7043 {\let\bbbl@0L@textsuperscript\@textsuperscript
7044 \bbbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7045 \let\bbbl@latinarabic=\@arabic
7046 \let\bbbl@0L@arabic\@arabic
7047 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
7048 \ifpackagewith{babel}{bidi=default}%
7049 {\let\bbbl@asciroman=\@roman
7050 \let\bbbl@0L@roman\@roman
7051 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
7052 \let\bbbl@asciiRoman=\@Roman
7053 \let\bbbl@0L@roman\@Roman
7054 \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
7055 \let\bbbl@0L@labelenumii\labelenumii
7056 \def\labelenumii{\theenumii}%
7057 \let\bbbl@0L@p@enumiii\p@enumiii
7058 \def\p@enumiii{\p@enumii}\theenumii{}}{}{}
7059 <@Footnote changes@>

```

```

7060 \IfBabelLayout{footnotes}%
7061   {\let\bbl@OL@footnote\footnote
7062     \BabelFootnote\footnote\language\language}%
7063     \BabelFootnote\localfootnote\language\language}%
7064     \BabelFootnote\mainfootnote\language\language}%
7065   {}

```

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

```

7066 \IfBabelLayout{extras}%
7067   {\bbl@ncarg\let\bbl@OL@underline\underscore}%
7068     \bbl@carg\bbl@sreplace\underscore}%
7069     {\$@@underline}\bgroup\bbl@nextfake$@@underline}%
7070     \bbl@carg\bbl@sreplace\underscore}%
7071     {\m@th$\m@th$\egroup}%
7072     \let\bbl@OL@LaTeXe\LaTeXe
7073     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7074       \if b\expandafter\car\@series\@nil\boldmath\fi
7075       \babelsublr}%
7076       \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}}}%
7077   {}
7078 \end{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.

```

7079 (*transforms)
7080 Babel.linebreaking.replacements = {}
7081 Babel.linebreaking.replacements[0] = {} -- pre
7082 Babel.linebreaking.replacements[1] = {} -- post
7083
7084 function Babel.tovalue(v)
7085   if type(v) == 'table' then
7086     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7087   else
7088     return v
7089   end
7090 end
7091
7092 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7093
7094 function Babel.set_hboxed(head, gc)
7095   for item in node.traverse(head) do
7096     node.set_attribute(item, Babel.attr_hboxed, 1)
7097   end
7098   return head
7099 end
7100
7101 Babel.fetch_subtext = {}
7102
7103 Babel.ignore_pre_char = function(node)
7104   return (node.lang == Babel.nohyphenation)
7105 end

```

```

7106
7107 Babel.show_transforms = false
7108
7109 -- Merging both functions doesn't seem feasible, because there are too
7110 -- many differences.
7111 Babel.fetch_subtext[0] = function(head)
7112   local word_string = ''
7113   local word_nodes = {}
7114   local lang
7115   local item = head
7116   local inmath = false
7117
7118   while item do
7119     if item.id == 11 then
7120       inmath = (item.subtype == 0)
7121     end
7122
7123     if inmath then
7124       -- pass
7125
7126     elseif item.id == 29 then
7127       local locale = node.get_attribute(item, Babel.attr_locale)
7128
7129       if lang == locale or lang == nil then
7130         lang = lang or locale
7131         if Babel.ignore_pre_char(item) then
7132           word_string = word_string .. Babel.us_char
7133         else
7134           if node.has_attribute(item, Babel.attr_hboxed) then
7135             word_string = word_string .. Babel.us_char
7136           else
7137             word_string = word_string .. unicode.utf8.char(item.char)
7138           end
7139         end
7140       end
7141       word_nodes[#word_nodes+1] = item
7142     else
7143       break
7144     end
7145
7146     elseif item.id == 12 and item.subtype == 13 then
7147       if node.has_attribute(item, Babel.attr_hboxed) then
7148         word_string = word_string .. Babel.us_char
7149       else
7150         word_string = word_string .. ' '
7151       end
7152       word_nodes[#word_nodes+1] = item
7153
7154       -- Ignore leading unrecognized nodes, too.
7155       elseif word_string ~= '' then
7156         word_string = word_string .. Babel.us_char
7157         word_nodes[#word_nodes+1] = item -- Will be ignored
7158       end
7159
7160     item = item.next
7161   end
7162
7163   -- Here and above we remove some trailing chars but not the
7164   -- corresponding nodes. But they aren't accessed.
7165   if word_string:sub(-1) == ' ' then
7166     word_string = word_string:sub(1,-2)
7167   end
7168   if Babel.show_transforms then texio.write_nl(word_string) end

```

```

7169 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7170 return word_string, word_nodes, item, lang
7171 end
7172
7173 Babel.fetch_subtext[1] = function(head)
7174   local word_string = ''
7175   local word_nodes = {}
7176   local lang
7177   local item = head
7178   local inmath = false
7179
7180   while item do
7181
7182     if item.id == 11 then
7183       inmath = (item.subtype == 0)
7184     end
7185
7186     if inmath then
7187       -- pass
7188
7189     elseif item.id == 29 then
7190       if item.lang == lang or lang == nil then
7191         lang = lang or item.lang
7192         if node.has_attribute(item, Babel.attr_hboxed) then
7193           word_string = word_string .. Babel.us_char
7194         elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7195           word_string = word_string .. Babel.us_char
7196         else
7197           word_string = word_string .. unicode.utf8.char(item.char)
7198         end
7199         word_nodes[#word_nodes+1] = item
7200       else
7201         break
7202       end
7203
7204     elseif item.id == 7 and item.subtype == 2 then
7205       if node.has_attribute(item, Babel.attr_hboxed) then
7206         word_string = word_string .. Babel.us_char
7207       else
7208         word_string = word_string .. '='
7209       end
7210       word_nodes[#word_nodes+1] = item
7211
7212     elseif item.id == 7 and item.subtype == 3 then
7213       if node.has_attribute(item, Babel.attr_hboxed) then
7214         word_string = word_string .. Babel.us_char
7215       else
7216         word_string = word_string .. '|'
7217       end
7218       word_nodes[#word_nodes+1] = item
7219
7220       -- (1) Go to next word if nothing was found, and (2) implicitly
7221       -- remove leading USs.
7222       elseif word_string == '' then
7223         -- pass
7224
7225       -- This is the responsible for splitting by words.
7226       elseif (item.id == 12 and item.subtype == 13) then
7227         break
7228
7229       else
7230         word_string = word_string .. Babel.us_char
7231         word_nodes[#word_nodes+1] = item -- Will be ignored

```

```

7232     end
7233
7234     item = item.next
7235 end
7236 if Babel.show_transforms then texio.write_nl(word_string) end
7237 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7238 return word_string, word_nodes, item, lang
7239 end
7240
7241 function Babel.pre_hyphenate_replace(head)
7242   Babel.hyphenate_replace(head, 0)
7243 end
7244
7245 function Babel.post_hyphenate_replace(head)
7246   Babel.hyphenate_replace(head, 1)
7247 end
7248
7249 Babel.us_char = string.char(31)
7250
7251 function Babel.hyphenate_replace(head, mode)
7252   local u = unicode.utf8
7253   local lbkr = Babel.linebreaking.replacements[mode]
7254   local tovalue = Babel.tovalue
7255
7256   local word_head = head
7257
7258   if Babel.show_transforms then
7259     texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7260   end
7261
7262   while true do -- for each subtext block
7263
7264     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7265
7266     if Babel.debug then
7267       print()
7268       print((mode == 0) and '====<' or '====>', w)
7269     end
7270
7271     if nw == nil and w == '' then break end
7272
7273     if not lang then goto next end
7274     if not lbkr[lang] then goto next end
7275
7276     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7277     -- loops are nested.
7278     for k=1, #lbkr[lang] do
7279       local p = lbkr[lang][k].pattern
7280       local r = lbkr[lang][k].replace
7281       local attr = lbkr[lang][k].attr or -1
7282
7283       if Babel.debug then
7284         print('*****', p, mode)
7285       end
7286
7287       -- This variable is set in some cases below to the first *byte*
7288       -- after the match, either as found by u.match (faster) or the
7289       -- computed position based on sc if w has changed.
7290       local last_match = 0
7291       local step = 0
7292
7293       -- For every match.
7294       while true do

```

```

7295     if Babel.debug then
7296         print('====')
7297     end
7298     local new -- used when inserting and removing nodes
7299     local dummy_node -- used by after
7300
7301     local matches = { u.match(w, p, last_match) }
7302
7303     if #matches < 2 then break end
7304
7305     -- Get and remove empty captures (with ()'s, which return a
7306     -- number with the position), and keep actual captures
7307     -- (from (...)), if any, in matches.
7308     local first = table.remove(matches, 1)
7309     local last = table.remove(matches, #matches)
7310     -- Non re-fetched substrings may contain \31, which separates
7311     -- subsubstrings.
7312     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7313
7314     local save_last = last -- with A()BC()D, points to D
7315
7316     -- Fix offsets, from bytes to unicode. Explained above.
7317     first = u.len(w:sub(1, first-1)) + 1
7318     last = u.len(w:sub(1, last-1)) -- now last points to C
7319
7320     -- This loop stores in a small table the nodes
7321     -- corresponding to the pattern. Used by 'data' to provide a
7322     -- predictable behavior with 'insert' (w_nodes is modified on
7323     -- the fly), and also access to 'remove'd nodes.
7324     local sc = first-1 -- Used below, too
7325     local data_nodes = {}
7326
7327     local enabled = true
7328     for q = 1, last-first+1 do
7329         data_nodes[q] = w_nodes[sc+q]
7330         if enabled
7331             and attr > -1
7332             and not node.has_attribute(data_nodes[q], attr)
7333         then
7334             enabled = false
7335         end
7336     end
7337
7338     -- This loop traverses the matched substring and takes the
7339     -- corresponding action stored in the replacement list.
7340     -- sc = the position in substr nodes / string
7341     -- rc = the replacement table index
7342     local rc = 0
7343
7344     ----- TODO. dummy_node?
7345     while rc < last-first+1 or dummy_node do -- for each replacement
7346         if Babel.debug then
7347             print('.....', rc + 1)
7348         end
7349         sc = sc + 1
7350         rc = rc + 1
7351
7352         if Babel.debug then
7353             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7354             local ss = ''
7355             for itt in node.traverse(head) do
7356                 if itt.id == 29 then
7357                     ss = ss .. unicode.utf8.char(itt.char)

```

```

7358         else
7359             ss = ss .. '{' .. itt.id .. '}'
7360         end
7361     end
7362     print('*****', ss)
7363
7364 end
7365
7366 local crep = r[rc]
7367 local item = w_nodes[sc]
7368 local item_base = item
7369 local placeholder = Babel.us_char
7370 local d
7371
7372 if crep and crep.data then
7373     item_base = data_nodes[crep.data]
7374 end
7375
7376 if crep then
7377     step = crep.step or step
7378 end
7379
7380 if crep and crep.after then
7381     crep.insert = true
7382     if dummy_node then
7383         item = dummy_node
7384     else -- TODO. if there is a node after?
7385         d = node.copy(item_base)
7386         head, item = node.insert_after(head, item, d)
7387         dummy_node = item
7388     end
7389 end
7390
7391 if crep and not crep.after and dummy_node then
7392     node.remove(head, dummy_node)
7393     dummy_node = nil
7394 end
7395
7396 if not enabled then
7397     last_match = save_last
7398     goto next
7399
7400 elseif crep and next(crep) == nil then -- = {}
7401     if step == 0 then
7402         last_match = save_last -- Optimization
7403     else
7404         last_match = utf8.offset(w, sc+step)
7405     end
7406     goto next
7407
7408 elseif crep == nil or crep.remove then
7409     node.remove(head, item)
7410     table.remove(w_nodes, sc)
7411     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7412     sc = sc - 1 -- Nothing has been inserted.
7413     last_match = utf8.offset(w, sc+1+step)
7414     goto next
7415
7416 elseif crep and crep.kashida then -- Experimental
7417     node.set_attribute(item,
7418         Babel.attr_kashida,
7419         crep.kashida)
7420     last_match = utf8.offset(w, sc+1+step)

```

```

7421         goto next
7422
7423     elseif crep and crep.string then
7424         local str = crep.string(matches)
7425         if str == '' then -- Gather with nil
7426             node.remove(head, item)
7427             table.remove(w_nodes, sc)
7428             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7429             sc = sc - 1 -- Nothing has been inserted.
7430         else
7431             local loop_first = true
7432             for s in string.utfvalues(str) do
7433                 d = node.copy(item_base)
7434                 d.char = s
7435                 if loop_first then
7436                     loop_first = false
7437                     head, new = node.insert_before(head, item, d)
7438                     if sc == 1 then
7439                         word_head = head
7440                     end
7441                     w_nodes[sc] = d
7442                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7443                 else
7444                     sc = sc + 1
7445                     head, new = node.insert_before(head, item, d)
7446                     table.insert(w_nodes, sc, new)
7447                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7448                 end
7449                 if Babel.debug then
7450                     print('.....', 'str')
7451                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7452                 end
7453             end -- for
7454             node.remove(head, item)
7455         end -- if ''
7456         last_match = utf8.offset(w, sc+1+step)
7457         goto next
7458
7459     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7460         d = node.new(7, 3) -- (disc, regular)
7461         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7462         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7463         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7464         d.attr = item_base.attr
7465         if crep.pre == nil then -- TeXbook p96
7466             d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7467         else
7468             d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7469         end
7470         placeholder = '|'
7471         head, new = node.insert_before(head, item, d)
7472
7473     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7474         -- ERROR
7475
7476     elseif crep and crep.penalty then
7477         d = node.new(14, 0) -- (penalty, userpenalty)
7478         d.attr = item_base.attr
7479         d.penalty = tovalue(crep.penalty)
7480         head, new = node.insert_before(head, item, d)
7481
7482     elseif crep and crep.space then
7483         -- 655360 = 10 pt = 10 * 65536 sp

```



```

7484         d = node.new(12, 13)          -- (glue, spaceskip)
7485         local quad = font.getfont(item_base.font).size or 655360
7486         node.setglue(d, tovalue(crep.space[1]) * quad,
7487                     tovalue(crep.space[2]) * quad,
7488                     tovalue(crep.space[3]) * quad)
7489         if mode == 0 then
7490             placeholder = ' '
7491         end
7492         head, new = node.insert_before(head, item, d)
7493
7494     elseif crep and crep.norule then
7495         -- 655360 = 10 pt = 10 * 65536 sp
7496         d = node.new(2, 3)          -- (rule, empty) = \no*rule
7497         local quad = font.getfont(item_base.font).size or 655360
7498         d.width  = tovalue(crep.norule[1]) * quad
7499         d.height = tovalue(crep.norule[2]) * quad
7500         d.depth  = tovalue(crep.norule[3]) * quad
7501         head, new = node.insert_before(head, item, d)
7502
7503     elseif crep and crep.spacefactor then
7504         d = node.new(12, 13)          -- (glue, spaceskip)
7505         local base_font = font.getfont(item_base.font)
7506         node.setglue(d,
7507                     tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7508                     tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7509                     tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7510         if mode == 0 then
7511             placeholder = ' '
7512         end
7513         head, new = node.insert_before(head, item, d)
7514
7515     elseif mode == 0 and crep and crep.space then
7516         -- ERROR
7517
7518     elseif crep and crep.kern then
7519         d = node.new(13, 1)          -- (kern, user)
7520         local quad = font.getfont(item_base.font).size or 655360
7521         d.attr = item_base.attr
7522         d.kern = tovalue(crep.kern) * quad
7523         head, new = node.insert_before(head, item, d)
7524
7525     elseif crep and crep.node then
7526         d = node.new(crep.node[1], crep.node[2])
7527         d.attr = item_base.attr
7528         head, new = node.insert_before(head, item, d)
7529
7530     end -- i.e., replacement cases
7531
7532     -- Shared by disc, space(factor), kern, node and penalty.
7533     if sc == 1 then
7534         word_head = head
7535     end
7536     if crep.insert then
7537         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7538         table.insert(w_nodes, sc, new)
7539         last = last + 1
7540     else
7541         w_nodes[sc] = d
7542         node.remove(head, item)
7543         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7544     end
7545
7546     last_match = utf8.offset(w, sc+1+step)

```

```

7547
7548         ::next::
7549
7550     end -- for each replacement
7551
7552     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7553     if Babel.debug then
7554         print('.....', '/')
7555         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7556     end
7557
7558     if dummy_node then
7559         node.remove(head, dummy_node)
7560         dummy_node = nil
7561     end
7562
7563     end -- for match
7564
7565 end -- for patterns
7566
7567     ::next::
7568     word_head = nw
7569 end -- for substring
7570
7571 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7572 return head
7573 end
7574
7575 -- This table stores capture maps, numbered consecutively
7576 Babel.capture_maps = {}
7577
7578 -- The following functions belong to the next macro
7579 function Babel.capture_func(key, cap)
7580     local ret = "[" .. cap:gsub('{{[0-9]}}', "")..m[%1]..["] .. "]"
7581     local cnt
7582     local u = unicode.utf8
7583     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(\.\.)}', Babel.capture_func_map)
7584     if cnt == 0 then
7585         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
7586             function (n)
7587                 return u.char(tonumber(n, 16))
7588             end)
7589     end
7590     ret = ret:gsub("%[%]%.%", '')
7591     ret = ret:gsub("%.%[%]%", '')
7592     return key .. [=function(m) return ] .. ret .. [=end]]
7593 end
7594
7595 function Babel.capt_map(from, mapno)
7596     return Babel.capture_maps[mapno][from] or from
7597 end
7598
7599 -- Handle the {n|abc|ABC} syntax in captures
7600 function Babel.capture_func_map(capno, from, to)
7601     local u = unicode.utf8
7602     from = u.gsub(from, '{{(%x%x%x%x+)}}',
7603         function (n)
7604             return u.char(tonumber(n, 16))
7605         end)
7606     to = u.gsub(to, '{{(%x%x%x%x+)}}',
7607         function (n)
7608             return u.char(tonumber(n, 16))
7609         end)

```

```

7610 local froms = {}
7611 for s in string.utfcharacters(from) do
7612     table.insert(froms, s)
7613 end
7614 local cnt = 1
7615 table.insert(Babel.capture_maps, {})
7616 local mlen = table.getn(Babel.capture_maps)
7617 for s in string.utfcharacters(to) do
7618     Babel.capture_maps[mlen][froms[cnt]] = s
7619     cnt = cnt + 1
7620 end
7621 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7622     (mlen) .. ")]" .. "["
7623 end
7624
7625 -- Create/Extend reversed sorted list of kashida weights:
7626 function Babel.capture_kashida(key, wt)
7627     wt = tonumber(wt)
7628     if Babel.kashida_wts then
7629         for p, q in ipairs(Babel.kashida_wts) do
7630             if wt == q then
7631                 break
7632             elseif wt > q then
7633                 table.insert(Babel.kashida_wts, p, wt)
7634                 break
7635             elseif table.getn(Babel.kashida_wts) == p then
7636                 table.insert(Babel.kashida_wts, wt)
7637             end
7638         end
7639     else
7640         Babel.kashida_wts = { wt }
7641     end
7642     return 'kashida = ' .. wt
7643 end
7644
7645 function Babel.capture_node(id, subtype)
7646     local sbt = 0
7647     for k, v in pairs(node.subtypes(id)) do
7648         if v == subtype then sbt = k end
7649     end
7650     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7651 end
7652
7653 -- Experimental: applies prehyphenation transforms to a string (letters
7654 -- and spaces).
7655 function Babel.string_prehyphenation(str, locale)
7656     local n, head, last, res
7657     head = node.new(8, 0) -- dummy (hack just to start)
7658     last = head
7659     for s in string.utfvalues(str) do
7660         if s == 20 then
7661             n = node.new(12, 0)
7662         else
7663             n = node.new(29, 0)
7664             n.char = s
7665         end
7666         node.set_attribute(n, Babel.attr_locale, locale)
7667         last.next = n
7668         last = n
7669     end
7670     head = Babel.hyphenate_replace(head, 0)
7671     res = ''
7672     for n in node.traverse(head) do

```

```

7673     if n.id == 12 then
7674         res = res .. ' '
7675     elseif n.id == 29 then
7676         res = res .. unicode.utf8.char(n.char)
7677     end
7678 end
7679 tex.print(res)
7680 end
7681 /transforms

```

10.14 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

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

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

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

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

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

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

```

7682 (*basic-r)
7683 Babel.bidi_enabled = true
7684
7685 require('babel-data-bidi.lua')
7686
7687 local characters = Babel.characters
7688 local ranges = Babel.ranges
7689
7690 local DIR = node.id("dir")
7691
7692 local function dir_mark(head, from, to, outer)
7693     dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7694     local d = node.new(DIR)
7695     d.dir = '+' .. dir

```

```

7696 node.insert_before(head, from, d)
7697 d = node.new(DIR)
7698 d.dir = '-' .. dir
7699 node.insert_after(head, to, d)
7700 end
7701
7702 function Babel.bidi(head, ispar)
7703   local first_n, last_n          -- first and last char with nums
7704   local last_es                  -- an auxiliary 'last' used with nums
7705   local first_d, last_d          -- first and last char in L/R block
7706   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):

```

```

7707   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7708   local strong_lr = (strong == 'l') and 'l' or 'r'
7709   local outer = strong
7710
7711   local new_dir = false
7712   local first_dir = false
7713   local inmath = false
7714
7715   local last_lr
7716
7717   local type_n = ''
7718
7719   for item in node.traverse(head) do
7720
7721     -- three cases: glyph, dir, otherwise
7722     if item.id == node.id'glyph'
7723       or (item.id == 7 and item.subtype == 2) then
7724
7725       local itemchar
7726       if item.id == 7 and item.subtype == 2 then
7727         itemchar = item.replace.char
7728       else
7729         itemchar = item.char
7730       end
7731       local chardata = characters[itemchar]
7732       dir = chardata and chardata.d or nil
7733       if not dir then
7734         for nn, et in ipairs(ranges) do
7735           if itemchar < et[1] then
7736             break
7737           elseif itemchar <= et[2] then
7738             dir = et[3]
7739             break
7740           end
7741         end
7742       end
7743       dir = dir or 'l'
7744       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).

```

7745   if new_dir then
7746     attr_dir = 0
7747     for at in node.traverse(item.attr) do
7748       if at.number == Babel.attr_dir then
7749         attr_dir = at.value & 0x3

```

```

7750         end
7751     end
7752     if attr_dir == 1 then
7753         strong = 'r'
7754     elseif attr_dir == 2 then
7755         strong = 'al'
7756     else
7757         strong = 'l'
7758     end
7759     strong_lr = (strong == 'l') and 'l' or 'r'
7760     outer = strong_lr
7761     new_dir = false
7762 end
7763
7764 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

7765     dir_real = dir          -- We need dir_real to set strong below
7766     if dir == 'al' then dir = 'r' end -- W3

```

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

```

7767     if strong == 'al' then
7768         if dir == 'en' then dir = 'an' end          -- W2
7769         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7770         strong_lr = 'r'          -- W3
7771     end

```

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

```

7772     elseif item.id == node.id'dir' and not inmath then
7773         new_dir = true
7774         dir = nil
7775     elseif item.id == node.id'math' then
7776         inmath = (item.subtype == 0)
7777     else
7778         dir = nil          -- Not a char
7779     end

```

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

```

7780     if dir == 'en' or dir == 'an' or dir == 'et' then
7781         if dir ~= 'et' then
7782             type_n = dir
7783         end
7784         first_n = first_n or item
7785         last_n = last_es or item
7786         last_es = nil
7787     elseif dir == 'es' and last_n then -- W3+W6
7788         last_es = item
7789     elseif dir == 'cs' then          -- it's right - do nothing
7790     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7791         if strong_lr == 'r' and type_n ~= '' then
7792             dir_mark(head, first_n, last_n, 'r')
7793         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7794             dir_mark(head, first_n, last_n, 'r')
7795             dir_mark(head, first_d, last_d, outer)
7796             first_d, last_d = nil, nil
7797         elseif strong_lr == 'l' and type_n ~= '' then
7798             last_d = last_n
7799         end
7800         type_n = ''

```

```

7801     first_n, last_n = nil, nil
7802 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:

```

7803     if dir == 'l' or dir == 'r' then
7804         if dir ~= outer then
7805             first_d = first_d or item
7806             last_d = item
7807         elseif first_d and dir ~= strong_lr then
7808             dir_mark(head, first_d, last_d, outer)
7809             first_d, last_d = nil, nil
7810         end
7811     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.

```

7812     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7813         item.char = characters[item.char] and
7814             characters[item.char].m or item.char
7815     elseif (dir or new_dir) and last_lr ~= item then
7816         local mir = outer .. strong_lr .. (dir or outer)
7817         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7818             for ch in node.traverse(node.next(last_lr)) do
7819                 if ch == item then break end
7820                 if ch.id == node.id'glyph' and characters[ch.char] then
7821                     ch.char = characters[ch.char].m or ch.char
7822                 end
7823             end
7824         end
7825     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).

```

7826     if dir == 'l' or dir == 'r' then
7827         last_lr = item
7828         strong = dir_real          -- Don't search back - best save now
7829         strong_lr = (strong == 'l') and 'l' or 'r'
7830     elseif new_dir then
7831         last_lr = nil
7832     end
7833 end

```

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

```

7834     if last_lr and outer == 'r' then
7835         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7836             if characters[ch.char] then
7837                 ch.char = characters[ch.char].m or ch.char
7838             end
7839         end
7840     end
7841     if first_n then
7842         dir_mark(head, first_n, last_n, outer)
7843     end
7844     if first_d then
7845         dir_mark(head, first_d, last_d, outer)
7846     end

```

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

```
7847 return node.prev(head) or head
7848 end
7849 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
7850 ⟨*basic⟩
7851 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7852
7853 Babel.fontmap = Babel.fontmap or {}
7854 Babel.fontmap[0] = {}      -- l
7855 Babel.fontmap[1] = {}      -- r
7856 Babel.fontmap[2] = {}      -- al/an
7857
7858 -- To cancel mirroring. Also OML, OMS, U?
7859 Babel.symbol_fonts = Babel.symbol_fonts or {}
7860 Babel.symbol_fonts[font.id('tenln')] = true
7861 Babel.symbol_fonts[font.id('tenlnw')] = true
7862 Babel.symbol_fonts[font.id('tencirc')] = true
7863 Babel.symbol_fonts[font.id('tencircw')] = true
7864
7865 Babel.bidi_enabled = true
7866 Babel.mirroring_enabled = true
7867
7868 require('babel-data-bidi.lua')
7869
7870 local characters = Babel.characters
7871 local ranges = Babel.ranges
7872
7873 local DIR = node.id('dir')
7874 local GLYPH = node.id('glyph')
7875
7876 local function insert_implicit(head, state, outer)
7877   local new_state = state
7878   if state.sim and state.eim and state.sim ~= state.eim then
7879     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7880     local d = node.new(DIR)
7881     d.dir = '+' .. dir
7882     node.insert_before(head, state.sim, d)
7883     local d = node.new(DIR)
7884     d.dir = '-' .. dir
7885     node.insert_after(head, state.eim, d)
7886   end
7887   new_state.sim, new_state.eim = nil, nil
7888   return head, new_state
7889 end
7890
7891 local function insert_numeric(head, state)
7892   local new
7893   local new_state = state
7894   if state.san and state.ean and state.san ~= state.ean then
7895     local d = node.new(DIR)
7896     d.dir = '+TLT'
7897     _, new = node.insert_before(head, state.san, d)
7898     if state.san == state.sim then state.sim = new end
7899     local d = node.new(DIR)
7900     d.dir = '-TLT'
7901     _, new = node.insert_after(head, state.ean, d)
7902     if state.ean == state.eim then state.eim = new end
7903   end
7904   new_state.san, new_state.ean = nil, nil
7905   return head, new_state
```



```

7906 end
7907
7908 local function glyph_not_symbol_font(node)
7909   if node.id == GLYPH then
7910     return not Babel.symbol_fonts[node.font]
7911   else
7912     return false
7913   end
7914 end
7915
7916 -- TODO - \hbox with an explicit dir can lead to wrong results
7917 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7918 -- was made to improve the situation, but the problem is the 3-dir
7919 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7920 -- well.
7921
7922 function Babel.bidi(head, ispar, hdir)
7923   local d    -- d is used mainly for computations in a loop
7924   local prev_d = ''
7925   local new_d = false
7926
7927   local nodes = {}
7928   local outer_first = nil
7929   local inmath = false
7930
7931   local glue_d = nil
7932   local glue_i = nil
7933
7934   local has_en = false
7935   local first_et = nil
7936
7937   local has_hyperlink = false
7938
7939   local ATDIR = Babel.attr_dir
7940   local attr_d, temp
7941   local locale_d
7942
7943   local save_outer
7944   local locale_d = node.get_attribute(head, ATDIR)
7945   if locale_d then
7946     locale_d = locale_d & 0x3
7947     save_outer = (locale_d == 0 and 'l') or
7948                  (locale_d == 1 and 'r') or
7949                  (locale_d == 2 and 'al')
7950   elseif ispar then -- Or error? Shouldn't happen
7951     -- when the callback is called, we are just _after_ the box,
7952     -- and the textdir is that of the surrounding text
7953     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7954   else -- Empty box
7955     save_outer = ('TRT' == hdir) and 'r' or 'l'
7956   end
7957   local outer = save_outer
7958   local last = outer
7959   -- 'al' is only taken into account in the first, current loop
7960   if save_outer == 'al' then save_outer = 'r' end
7961
7962   local fontmap = Babel.fontmap
7963
7964   for item in node.traverse(head) do
7965     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
7966     locale_d = node.get_attribute(item, ATDIR)
7967     node.set_attribute(item, ATDIR, 0x80)

```

```

7969
7970 -- In what follows, #node is the last (previous) node, because the
7971 -- current one is not added until we start processing the neutrals.
7972 -- three cases: glyph, dir, otherwise
7973 if glyph_not_symbol_font(item)
7974     or (item.id == 7 and item.subtype == 2) then
7975
7976     if locale_d == 0x80 then goto nextnode end
7977
7978     local d_font = nil
7979     local item_r
7980     if item.id == 7 and item.subtype == 2 then
7981         item_r = item.replace -- automatic discs have just 1 glyph
7982     else
7983         item_r = item
7984     end
7985
7986     local chardata = characters[item_r.char]
7987     d = chardata and chardata.d or nil
7988     if not d or d == 'nsm' then
7989         for nn, et in ipairs(ranges) do
7990             if item_r.char < et[1] then
7991                 break
7992             elseif item_r.char <= et[2] then
7993                 if not d then d = et[3]
7994                 elseif d == 'nsm' then d_font = et[3]
7995                 end
7996                 break
7997             end
7998         end
7999     end
8000     d = d or 'l'
8001
8002     -- A short 'pause' in bidi for mapfont
8003     -- %%% TODO. move if fontmap here
8004     d_font = d_font or d
8005     d_font = (d_font == 'l' and 0) or
8006             (d_font == 'nsm' and 0) or
8007             (d_font == 'r' and 1) or
8008             (d_font == 'al' and 2) or
8009             (d_font == 'an' and 2) or nil
8010     if d_font and fontmap and fontmap[d_font][item_r.font] then
8011         item_r.font = fontmap[d_font][item_r.font]
8012     end
8013
8014     if new_d then
8015         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8016         if inmath then
8017             attr_d = 0
8018         else
8019             attr_d = locale_d & 0x3
8020         end
8021         if attr_d == 1 then
8022             outer_first = 'r'
8023             last = 'r'
8024         elseif attr_d == 2 then
8025             outer_first = 'r'
8026             last = 'al'
8027         else
8028             outer_first = 'l'
8029             last = 'l'
8030         end
8031         outer = last

```

```

8032     has_en = false
8033     first_et = nil
8034     new_d = false
8035 end
8036
8037 if glue_d then
8038     if (d == 'l' and 'l' or 'r') ~= glue_d then
8039         table.insert(nodes, {glue_i, 'on', nil})
8040     end
8041     glue_d = nil
8042     glue_i = nil
8043 end
8044
8045 elseif item.id == DIR then
8046     d = nil
8047     new_d = true
8048
8049 elseif item.id == node.id'glue' and item.subtype == 13 then
8050     glue_d = d
8051     glue_i = item
8052     d = nil
8053
8054 elseif item.id == node.id'math' then
8055     inmath = (item.subtype == 0)
8056
8057 elseif item.id == 8 and item.subtype == 19 then
8058     has_hyperlink = true
8059
8060 else
8061     d = nil
8062 end
8063
8064 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8065 if last == 'al' and d == 'en' then
8066     d = 'an'           -- W3
8067 elseif last == 'al' and (d == 'et' or d == 'es') then
8068     d = 'on'           -- W6
8069 end
8070
8071 -- EN + CS/ES + EN      -- W4
8072 if d == 'en' and #nodes >= 2 then
8073     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8074         and nodes[#nodes-1][2] == 'en' then
8075         nodes[#nodes][2] = 'en'
8076     end
8077 end
8078
8079 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
8080 if d == 'an' and #nodes >= 2 then
8081     if (nodes[#nodes][2] == 'cs')
8082         and nodes[#nodes-1][2] == 'an' then
8083         nodes[#nodes][2] = 'an'
8084     end
8085 end
8086
8087 -- ET/EN                -- W5 + W7->l / W6->on
8088 if d == 'et' then
8089     first_et = first_et or (#nodes + 1)
8090 elseif d == 'en' then
8091     has_en = true
8092     first_et = first_et or (#nodes + 1)
8093 elseif first_et then    -- d may be nil here !
8094     if has_en then

```

```

8095         if last == 'l' then
8096             temp = 'l'      -- W7
8097         else
8098             temp = 'en'     -- W5
8099         end
8100     else
8101         temp = 'on'         -- W6
8102     end
8103     for e = first_et, #nodes do
8104         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8105     end
8106     first_et = nil
8107     has_en = false
8108 end
8109
8110 -- Force mathdir in math if ON (currently works as expected only
8111 -- with 'l')
8112
8113 if inmath and d == 'on' then
8114     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8115 end
8116
8117 if d then
8118     if d == 'al' then
8119         d = 'r'
8120         last = 'al'
8121     elseif d == 'l' or d == 'r' then
8122         last = d
8123     end
8124     prev_d = d
8125     table.insert(nodes, {item, d, outer_first})
8126 end
8127
8128 outer_first = nil
8129
8130 ::nextnode::
8131
8132 end -- for each node
8133
8134 -- TODO -- repeated here in case EN/ET is the last node. Find a
8135 -- better way of doing things:
8136 if first_et then      -- dir may be nil here !
8137     if has_en then
8138         if last == 'l' then
8139             temp = 'l'      -- W7
8140         else
8141             temp = 'en'     -- W5
8142         end
8143     else
8144         temp = 'on'         -- W6
8145     end
8146     for e = first_et, #nodes do
8147         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8148     end
8149 end
8150
8151 -- dummy node, to close things
8152 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8153
8154 ----- NEUTRAL -----
8155
8156 outer = save_outer
8157 last = outer

```

```

8158
8159 local first_on = nil
8160
8161 for q = 1, #nodes do
8162     local item
8163
8164     local outer_first = nodes[q][3]
8165     outer = outer_first or outer
8166     last = outer_first or last
8167
8168     local d = nodes[q][2]
8169     if d == 'an' or d == 'en' then d = 'r' end
8170     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8171
8172     if d == 'on' then
8173         first_on = first_on or q
8174     elseif first_on then
8175         if last == d then
8176             temp = d
8177         else
8178             temp = outer
8179         end
8180         for r = first_on, q - 1 do
8181             nodes[r][2] = temp
8182             item = nodes[r][1] -- MIRRORING
8183             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8184                 and temp == 'r' and characters[item.char] then
8185                 local font_mode = ''
8186                 if item.font > 0 and font.fonts[item.font].properties then
8187                     font_mode = font.fonts[item.font].properties.mode
8188                 end
8189                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8190                     item.char = characters[item.char].m or item.char
8191                 end
8192             end
8193         end
8194         first_on = nil
8195     end
8196
8197     if d == 'r' or d == 'l' then last = d end
8198 end
8199
8200 ----- IMPLICIT, REORDER -----
8201
8202 outer = save_outer
8203 last = outer
8204
8205 local state = {}
8206 state.has_r = false
8207
8208 for q = 1, #nodes do
8209
8210     local item = nodes[q][1]
8211
8212     outer = nodes[q][3] or outer
8213
8214     local d = nodes[q][2]
8215
8216     if d == 'nsm' then d = last end -- W1
8217     if d == 'en' then d = 'an' end
8218     local isdir = (d == 'r' or d == 'l')
8219
8220     if outer == 'l' and d == 'an' then

```

```

8221     state.san = state.san or item
8222     state.ean = item
8223 elseif state.san then
8224     head, state = insert_numeric(head, state)
8225 end
8226
8227 if outer == 'l' then
8228     if d == 'an' or d == 'r' then      -- im -> implicit
8229         if d == 'r' then state.has_r = true end
8230         state.sim = state.sim or item
8231         state.eim = item
8232     elseif d == 'l' and state.sim and state.has_r then
8233         head, state = insert_implicit(head, state, outer)
8234     elseif d == 'l' then
8235         state.sim, state.eim, state.has_r = nil, nil, false
8236     end
8237 else
8238     if d == 'an' or d == 'l' then
8239         if nodes[q][3] then -- nil except after an explicit dir
8240             state.sim = item -- so we move sim 'inside' the group
8241         else
8242             state.sim = state.sim or item
8243         end
8244         state.eim = item
8245     elseif d == 'r' and state.sim then
8246         head, state = insert_implicit(head, state, outer)
8247     elseif d == 'r' then
8248         state.sim, state.eim = nil, nil
8249     end
8250 end
8251
8252 if isdir then
8253     last = d      -- Don't search back - best save now
8254 elseif d == 'on' and state.san then
8255     state.san = state.san or item
8256     state.ean = item
8257 end
8258
8259 end
8260
8261 head = node.prev(head) or head
8262 % \end{macrocode}
8263 %
8264 % Now direction nodes has been distributed with relation to characters
8265 % and spaces, we need to take into account \TeX-specific elements in
8266 % the node list, to move them at an appropriate place. Firstly, with
8267 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8268 % that the latter are still discardable.
8269 %
8270 % \begin{macrocode}
8271 --- FIXES ---
8272 if has_hyperlink then
8273     local flag, linking = 0, 0
8274     for item in node.traverse(head) do
8275         if item.id == DIR then
8276             if item.dir == '+TRT' or item.dir == '+TLT' then
8277                 flag = flag + 1
8278             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8279                 flag = flag - 1
8280             end
8281         elseif item.id == 8 and item.subtype == 19 then
8282             linking = flag
8283         elseif item.id == 8 and item.subtype == 20 then

```

```

8284         if linking > 0 then
8285             if item.prev.id == DIR and
8286                 (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8287                 d = node.new(DIR)
8288                 d.dir = item.prev.dir
8289                 node.remove(head, item.prev)
8290                 node.insert_after(head, item, d)
8291             end
8292         end
8293         linking = 0
8294     end
8295 end
8296 end
8297
8298 for item in node.traverse_id(10, head) do
8299     local p = item
8300     local flag = false
8301     while p.prev and p.prev.id == 14 do
8302         flag = true
8303         p = p.prev
8304     end
8305     if flag then
8306         node.insert_before(head, p, node.copy(item))
8307         node.remove(head, item)
8308     end
8309 end
8310
8311 return head
8312 end
8313
8313 function Babel.unset_atdir(head)
8314     local ATDIR = Babel.attr_dir
8315     for item in node.traverse(head) do
8316         node.set_attribute(item, ATDIR, 0x80)
8317     end
8318     return head
8319 end
8320 /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.

```

8321 \*nil
8322 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8323 \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.

```
8324 \ifx\l@nil\@undefined
8325   \newlanguage\l@nil
8326   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
8327   \let\bbl@elt\relax
8328   \edef\bbl@languages{% Add it to the list of languages
8329     \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
8330 \fi
```

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

```
8331 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil **\datenil**

```
8332 \let\captionnil\@empty
8333 \let\datenil\@empty
```

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

```
8334 \def\bbl@inidata@nil{%
8335   \bbl@elt{identification}{tag.ini}{und}%
8336   \bbl@elt{identification}{load.level}{0}%
8337   \bbl@elt{identification}{charset}{utf8}%
8338   \bbl@elt{identification}{version}{1.0}%
8339   \bbl@elt{identification}{date}{2022-05-16}%
8340   \bbl@elt{identification}{name.local}{nil}%
8341   \bbl@elt{identification}{name.english}{nil}%
8342   \bbl@elt{identification}{name.babel}{nil}%
8343   \bbl@elt{identification}{tag.bcp47}{und}%
8344   \bbl@elt{identification}{language.tag.bcp47}{und}%
8345   \bbl@elt{identification}{tag.opentype}{dflt}%
8346   \bbl@elt{identification}{script.name}{Latin}%
8347   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8348   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8349   \bbl@elt{identification}{level}{1}%
8350   \bbl@elt{identification}{encodings}{}%
8351   \bbl@elt{identification}{derivate}{no}}
8352 \@namedef{bbl@tbcpl@nil}{und}
8353 \@namedef{bbl@lbcpl@nil}{und}
8354 \@namedef{bbl@casing@nil}{und} % TODO
8355 \@namedef{bbl@lotf@nil}{dflt}
8356 \@namedef{bbl@elname@nil}{nil}
8357 \@namedef{bbl@lname@nil}{nil}
8358 \@namedef{bbl@esname@nil}{Latin}
8359 \@namedef{bbl@sname@nil}{Latin}
8360 \@namedef{bbl@sbcpl@nil}{Latn}
8361 \@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.

```
8362 \ldf@finish{nil}
8363 \</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.


```

8364 << *Compute Julian day >> ≡
8365 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8366 \def\bbl@cs@gregleap#1{%
8367   (\bbl@fpmo{#1}{4} == 0) &&
8368   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
8369 \def\bbl@cs@jd#1#2#3{% year, month, day
8370   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8371     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8372     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8373     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8374 << /Compute Julian day >>

```

13.1. Islamic

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

```

8375 < *ca-islamic >
8376 \ExplSyntaxOn
8377 < @Compute Julian day >
8378 % == islamic (default)
8379 % Not yet implemented
8380 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8381 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8382   ((#3 + ceil(29.5 * (#2 - 1)) +
8383     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8384     1948439.5) - 1) }
8385 \namedef\bbl@ca@islamic-civil++{\bbl@ca@islamicvl@x{+2}}
8386 \namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl@x{+1}}
8387 \namedef\bbl@ca@islamic-civil{\bbl@ca@islamicvl@x{}}
8388 \namedef\bbl@ca@islamic-civil-{\bbl@ca@islamicvl@x{-1}}
8389 \namedef\bbl@ca@islamic-civil--{\bbl@ca@islamicvl@x{-2}}
8390 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8391   \edef\bbl@tempa{%
8392     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 } }%
8393   \edef#5{%
8394     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) } }%
8395   \edef#6{\fp_eval:n{
8396     min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) } }%
8397   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1 } }

```

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

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

```

8398 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8399 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8400 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8401 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8402 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8403 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8404 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8405 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8406 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8407 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8408 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8409 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8410 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8411 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8412 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8413 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8414 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8415 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%

```

```

8416 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8417 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8418 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8419 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8420 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8421 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8422 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8423 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8424 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8425 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8426 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8427 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8428 65401,65431,65460,65490,65520}
8429 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8430 \namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8431 \namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{0}}
8432 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8433 \ifnum#2>2014 \ifnum#2<2038
8434 \bbl@aferfi\expandafter\@gobble
8435 \fi\fi
8436 {\bbl@error{year-out-range}{2014-2038}{}}}%
8437 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8438 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8439 \count@\@ne
8440 \bbl@foreach\bbl@cs@umalqura@data{%
8441 \advance\count@\@ne
8442 \ifnum##1>\bbl@tempd\else
8443 \edef\bbl@tempe{\the\count@}%
8444 \edef\bbl@tempb{##1}%
8445 \fi}%
8446 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8447 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8448 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8449 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8450 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8451 \ExplSyntaxOff
8452 \bbl@add\bbl@precalendar{%
8453 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8454 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8455 \bbl@replace\bbl@ld@calendar{+}{}}%
8456 \bbl@replace\bbl@ld@calendar{-}{}}%
8457 </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 `hebcsl.sty`

```

8458 <*ca-hebrew>
8459 \newcount\bbl@cntcommon
8460 \def\bbl@remainder#1#2#3{%
8461 #3=#1\relax
8462 \divide #3 by #2\relax
8463 \multiply #3 by -#2\relax
8464 \advance #3 by #1\relax}%
8465 \newif\ifbbl@divisible
8466 \def\bbl@checkifdivisible#1#2{%
8467 {\countdef\tmp=0
8468 \bbl@remainder{#1}{#2}{\tmp}%
8469 \ifnum \tmp=0
8470 \global\bbl@divisibletrue
8471 \else
8472 \global\bbl@divisiblefalse

```

```

8473 \fi}}
8474 \newif\ifbbl@gregleap
8475 \def\bbl@ifgregleap#1{%
8476 \bbl@checkifdivisible{#1}{4}%
8477 \ifbbl@divisible
8478 \bbl@checkifdivisible{#1}{100}%
8479 \ifbbl@divisible
8480 \bbl@checkifdivisible{#1}{400}%
8481 \ifbbl@divisible
8482 \bbl@gregleaptrue
8483 \else
8484 \bbl@gregleapfalse
8485 \fi
8486 \else
8487 \bbl@gregleaptrue
8488 \fi
8489 \else
8490 \bbl@gregleapfalse
8491 \fi
8492 \ifbbl@gregleap}
8493 \def\bbl@gregdayspriormonths#1#2#3{%
8494 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8495 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8496 \bbl@ifgregleap{#2}%
8497 \ifnum #1 > 2
8498 \advance #3 by 1
8499 \fi
8500 \fi
8501 \global\bbl@cntcommon=#3}%
8502 #3=\bbl@cntcommon}
8503 \def\bbl@gregdaysprioryears#1#2{%
8504 {\countdef\tmpc=4
8505 \countdef\tmpb=2
8506 \tmpb=#1\relax
8507 \advance \tmpb by -1
8508 \tmpc=\tmpb
8509 \multiply \tmpc by 365
8510 #2=\tmpc
8511 \tmpc=\tmpb
8512 \divide \tmpc by 4
8513 \advance #2 by \tmpc
8514 \tmpc=\tmpb
8515 \divide \tmpc by 100
8516 \advance #2 by -\tmpc
8517 \tmpc=\tmpb
8518 \divide \tmpc by 400
8519 \advance #2 by \tmpc
8520 \global\bbl@cntcommon=#2\relax}%
8521 #2=\bbl@cntcommon}
8522 \def\bbl@absfromgreg#1#2#3#4{%
8523 {\countdef\tmpd=0
8524 #4=#1\relax
8525 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8526 \advance #4 by \tmpd
8527 \bbl@gregdaysprioryears{#3}{\tmpd}%
8528 \advance #4 by \tmpd
8529 \global\bbl@cntcommon=#4\relax}%
8530 #4=\bbl@cntcommon}
8531 \newif\ifbbl@hebrleap
8532 \def\bbl@checkleaphebryear#1{%
8533 {\countdef\tmpa=0
8534 \countdef\tmpb=1
8535 \tmpa=#1\relax

```

```

8536 \multiply \tmpa by 7
8537 \advance \tmpa by 1
8538 \bbl@remainder{\tmpa}{19}{\tmpb}%
8539 \ifnum \tmpb < 7
8540 \global\bbl@hebrleaptrue
8541 \else
8542 \global\bbl@hebrleapfalse
8543 \fi}}
8544 \def\bbl@hebreleapsedmonths#1#2{%
8545 {\countdef\tmpa=0
8546 \countdef\tmpb=1
8547 \countdef\tmpc=2
8548 \tmpa=#1\relax
8549 \advance \tmpa by -1
8550 #2=\tmpa
8551 \divide #2 by 19
8552 \multiply #2 by 235
8553 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8554 \tmpc=\tmpb
8555 \multiply \tmpb by 12
8556 \advance #2 by \tmpb
8557 \multiply \tmpc by 7
8558 \advance \tmpc by 1
8559 \divide \tmpc by 19
8560 \advance #2 by \tmpc
8561 \global\bbl@cntcommon=#2}%
8562 #2=\bbl@cntcommon}
8563 \def\bbl@hebreleapseddays#1#2{%
8564 {\countdef\tmpa=0
8565 \countdef\tmpb=1
8566 \countdef\tmpc=2
8567 \bbl@hebreleapsedmonths{#1}{#2}%
8568 \tmpa=#2\relax
8569 \multiply \tmpa by 13753
8570 \advance \tmpa by 5604
8571 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8572 \divide \tmpa by 25920
8573 \multiply #2 by 29
8574 \advance #2 by 1
8575 \advance #2 by \tmpa
8576 \bbl@remainder{#2}{7}{\tmpa}%
8577 \ifnum \tmpc < 19440
8578 \ifnum \tmpc < 9924
8579 \else
8580 \ifnum \tmpa=2
8581 \bbl@checkleaphebyear{#1}% of a common year
8582 \ifbbl@hebrleap
8583 \else
8584 \advance #2 by 1
8585 \fi
8586 \fi
8587 \fi
8588 \ifnum \tmpc < 16789
8589 \else
8590 \ifnum \tmpa=1
8591 \advance #1 by -1
8592 \bbl@checkleaphebyear{#1}% at the end of leap year
8593 \ifbbl@hebrleap
8594 \advance #2 by 1
8595 \fi
8596 \fi
8597 \fi
8598 \else

```

```

8599     \advance #2 by 1
8600 \fi
8601 \bbl@remainder{#2}{7}{\tmpa}%
8602 \ifnum \tmpa=0
8603     \advance #2 by 1
8604 \else
8605     \ifnum \tmpa=3
8606         \advance #2 by 1
8607     \else
8608         \ifnum \tmpa=5
8609             \advance #2 by 1
8610         \fi
8611     \fi
8612 \fi
8613 \global\bbl@cntcommon=#2\relax}%
8614 #2=\bbl@cntcommon}
8615 \def\bbl@daysinhebrewyear#1#2{%
8616     {\countdef\tmpe=12
8617     \bbl@hebreleapseddays{#1}{\tmpe}%
8618     \advance #1 by 1
8619     \bbl@hebreleapseddays{#1}{#2}%
8620     \advance #2 by -\tmpe
8621     \global\bbl@cntcommon=#2}%
8622 #2=\bbl@cntcommon}
8623 \def\bbl@hebrdayspriormonths#1#2#3{%
8624     {\countdef\tmpf= 14
8625     #3=\ifcase #1
8626         0 \or
8627         0 \or
8628         30 \or
8629         59 \or
8630         89 \or
8631         118 \or
8632         148 \or
8633         148 \or
8634         177 \or
8635         207 \or
8636         236 \or
8637         266 \or
8638         295 \or
8639         325 \or
8640         400
8641     \fi
8642     \bbl@checkleaphebrewyear{#2}%
8643     \ifbbl@hebrleap
8644         \ifnum #1 > 6
8645             \advance #3 by 30
8646         \fi
8647     \fi
8648     \bbl@daysinhebrewyear{#2}{\tmpf}%
8649     \ifnum #1 > 3
8650         \ifnum \tmpf=353
8651             \advance #3 by -1
8652         \fi
8653         \ifnum \tmpf=383
8654             \advance #3 by -1
8655         \fi
8656     \fi
8657     \ifnum #1 > 2
8658         \ifnum \tmpf=355
8659             \advance #3 by 1
8660         \fi
8661         \ifnum \tmpf=385

```

```

8662         \advance #3 by 1
8663     \fi
8664 \fi
8665 \global\bbl@cntcommon=#3\relax}%
8666 #3=\bbl@cntcommon}
8667 \def\bbl@absfromhebr#1#2#3#4{%
8668     {#4=#1\relax
8669     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8670     \advance #4 by #1\relax
8671     \bbl@hebreleapseddays{#3}{#1}%
8672     \advance #4 by #1\relax
8673     \advance #4 by -1373429
8674     \global\bbl@cntcommon=#4\relax}%
8675 #4=\bbl@cntcommon}
8676 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8677     {\countdef\tmpx= 17
8678     \countdef\tmpy= 18
8679     \countdef\tmpz= 19
8680     #6=#3\relax
8681     \global\advance #6 by 3761
8682     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8683     \tmpz=1 \tmpy=1
8684     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8685     \ifnum \tmpx > #4\relax
8686         \global\advance #6 by -1
8687         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8688     \fi
8689     \advance #4 by -\tmpx
8690     \advance #4 by 1
8691     #5=#4\relax
8692     \divide #5 by 30
8693     \loop
8694         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8695         \ifnum \tmpx < #4\relax
8696             \advance #5 by 1
8697             \tmpy=\tmpx
8698         \repeat
8699     \global\advance #5 by -1
8700     \global\advance #4 by -\tmpy}}
8701 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8702 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8703 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8704     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8705     \bbl@hebrfromgreg
8706     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8707     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8708     \edef#4{\the\bbl@hebyear}%
8709     \edef#5{\the\bbl@hebrmonth}%
8710     \edef#6{\the\bbl@hebrday}}
8711 \</ca-hebrew>

```

13.3. Persian

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

```

8712 \<*ca-persian>
8713 \ExplSyntaxOn
8714 <@Compute Julian day@>
8715 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8716     2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}

```

```

8717 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8718 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8719 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8720 \bbl@afterfi\expandafter\@gobble
8721 \fi\fi
8722 {\bbl@error{year-out-range}{2013-2050}{}}}%
8723 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8724 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8725 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8726 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8727 \ifnum\bbl@tempc<\bbl@tempb
8728 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8729 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8730 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8731 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8732 \fi
8733 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8734 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8735 \edef#5{\fp_eval:n{% set Jalali month
8736 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8737 \edef#6{\fp_eval:n{% set Jalali day
8738 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
8739 \ExplSyntaxOff
8740 </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.

```

8741 <*ca-coptic>
8742 \ExplSyntaxOn
8743 <@Compute Julian day@>
8744 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8745 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8746 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8747 \edef#4{\fp_eval:n{%
8748 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8749 \edef\bbl@tempc{\fp_eval:n{%
8750 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8751 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8752 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8753 \ExplSyntaxOff
8754 </ca-coptic>
8755 <*ca-ethiopic>
8756 \ExplSyntaxOn
8757 <@Compute Julian day@>
8758 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8759 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8760 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8761 \edef#4{\fp_eval:n{%
8762 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8763 \edef\bbl@tempc{\fp_eval:n{%
8764 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8765 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8766 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8767 \ExplSyntaxOff
8768 </ca-ethiopic>

```

13.5. Buddhist

That's very simple.

```

8769 <*ca-buddhist>

```

```

8770 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8771 \edef#4{\number\numexpr#1+543\relax}%
8772 \edef#5{#2}%
8773 \edef#6{#3}}
8774 </ca-buddhist>
8775 %
8776 % \subsection{Chinese}
8777 %
8778 % Brute force, with the Julian day of first day of each month. The
8779 % table has been computed with the help of \textsf{python-lunardate} by
8780 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8781 % is 2015-2044.
8782 %
8783 % \begin{macrocode}
8784 <*ca-chinese>
8785 \ExplSyntaxOn
8786 <@Compute Julian day@>
8787 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8788 \edef\bbl@tempd{\fp_eval:n{%
8789 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8790 \count@ \z@
8791 \@tempcnta=2015
8792 \bbl@foreach\bbl@cs@chinese@data{%
8793 \ifnum##1>\bbl@tempd\else
8794 \advance\count@\@ne
8795 \ifnum\count@>12
8796 \count@\@ne
8797 \advance\@tempcnta\@ne\fi
8798 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8799 \ifin@
8800 \advance\count@\m@ne
8801 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8802 \else
8803 \edef\bbl@tempe{\the\count@}%
8804 \fi
8805 \edef\bbl@tempb{##1}%
8806 \fi}%
8807 \edef#4{\the\@tempcnta}%
8808 \edef#5{\bbl@tempe}%
8809 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8810 \def\bbl@cs@chinese@leap{%
8811 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8812 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8813 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8814 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8815 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8816 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8817 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8818 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8819 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8820 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8821 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8822 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8823 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8824 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8825 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8826 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8827 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8828 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8829 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8830 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8831 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8832 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%

```



```

8833 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8834 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8835 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8836 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8837 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8838 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8839 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8840 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8841 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8842 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8843 10896,10926,10956,10986,11015,11045,11074,11103}
8844 \ExplSyntaxOff
8845 </ca-chinese>

```

14. Support for Plain T_EX (plain.def)

14.1. Not renaming hyphen.tex

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

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

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

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

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

```

8846 <*\bplain | blplain>
8847 \catcode`\{=1 % left brace is begin-group character
8848 \catcode`\}=2 % right brace is end-group character
8849 \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).

```

8850 \openin 0 hyphen.cfg
8851 \ifeof0
8852 \else
8853   \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.

```

8854 \def\input #1 {%
8855   \let\input\input
8856   \a hyphen.cfg
8857   \let\input\input
8858 }
8859 \fi
8860 </bplain | blplain>

```

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

```

8861 <bplain>\a plain.tex
8862 <blplain>\a lplain.tex

```

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

```

8863 <bplain>\def\fmtname{babel-plain}
8864 <blplain>\def\fmtname{babel-lplain}

```

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

14.2. Emulating some \LaTeX features

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

```

8865 <<*Emulate LaTeX>> ≡
8866 \def\@empty{}
8867 \def\loadlocalcfg#1{%
8868   \openin0#1.cfg
8869   \ifeof0
8870     \closein0
8871   \else
8872     \closein0
8873     {\immediate\write16{*****}%
8874      \immediate\write16{* Local config file #1.cfg used}%
8875      \immediate\write16{*}%
8876     }
8877     \input #1.cfg\relax
8878   \fi
8879   \@endofldf}

```

14.3. General tools

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

```

8880 \long\def\@firstofone#1{#1}
8881 \long\def\@firstoftwo#1#2{#1}
8882 \long\def\@secondoftwo#1#2{#2}
8883 \def\@nnil{\@nil}
8884 \def\@gobbletwo#1#2{}
8885 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8886 \def\@star@or@long#1{%
8887   \@ifstar
8888   {\let\l@ngrel@x\relax#1}%
8889   {\let\l@ngrel@x\long#1}}
8890 \let\l@ngrel@x\relax
8891 \def\@car#1#2\@nil{#1}
8892 \def\@cdr#1#2\@nil{#2}
8893 \let\@typeset@protect\relax
8894 \let\protected@edef\edef
8895 \long\def\@gobble#1{}
8896 \edef\@backslashchar{\expandafter\@gobble\string\}
8897 \def\strip@prefix#1>{}
8898 \def\g@addto@macro#1#2{{%
8899   \toks@\expandafter{#1#2}%
8900   \xdef#1{\the\toks@}}}
8901 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8902 \def\@nameuse#1{\csname #1\endcsname}
8903 \def\@ifundefined#1{%
8904   \expandafter\ifx\csname#1\endcsname\relax
8905     \expandafter\@firstoftwo
8906   \else
8907     \expandafter\@secondoftwo
8908   \fi}
8909 \def\@expandtwoargs#1#2#3{%
8910   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8911 \def\zap@space#1 #2{%
8912   #1%

```

```

8913 \ifx#2\@empty\else\expandafter\zap@space\fi
8914 #2}
8915 \let\bbl@trace\@gobble
8916 \def\bbl@error#1{% Implicit #2#3#4
8917 \begingroup
8918 \catcode`\=0 \catcode`\==12 \catcode`\`=12
8919 \catcode`\^M=5 \catcode`\%=14
8920 \input errbabel.def
8921 \endgroup
8922 \bbl@error{#1}}
8923 \def\bbl@warning#1{%
8924 \begingroup
8925 \newlinechar=`^^J
8926 \def\{^^J(babel) }%
8927 \message{\{#1}%
8928 \endgroup}
8929 \let\bbl@infowarn\bbl@warning
8930 \def\bbl@info#1{%
8931 \begingroup
8932 \newlinechar=`^^J
8933 \def\{^^J}%
8934 \wlog{#1}%
8935 \endgroup}

```

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

```

8936 \ifx\@preamblecmds\undefined
8937 \def\@preamblecmds{}
8938 \fi
8939 \def\@onlypreamble#1{%
8940 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8941 \@preamblecmds\do#1}}
8942 \@onlypreamble\@onlypreamble

```

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

```

8943 \def\begindocument{%
8944 \@begindocumenthook
8945 \global\let\@begindocumenthook\@undefined
8946 \def\do##1{\global\let##1\@undefined}%
8947 \@preamblecmds
8948 \global\let\do\noexpand}
8949 \ifx\@begindocumenthook\undefined
8950 \def\@begindocumenthook{}
8951 \fi
8952 \@onlypreamble\@begindocumenthook
8953 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8954 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8955 \@onlypreamble\AtEndOfPackage
8956 \def\@endoflfd{}
8957 \@onlypreamble\@endoflfd
8958 \let\bbl@afterlang\@empty
8959 \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.

```

8960 \catcode`\&=\z@
8961 \ifx&\if@files\@undefined
8962 \expandafter\let\csname if@files\expandafter\endcsname
8963 \csname iffalse\endcsname

```

```

8964 \fi
8965 \catcode`\&=4

Mimic LATEX's commands to define control sequences.

8966 \def\newcommand{\@star@or@long\new@command}
8967 \def\new@command#1{%
8968   \@testopt{\@newcommand#1}0}
8969 \def\@newcommand#1[#2]{%
8970   \@ifnextchar [{\@xargdef#1[#2]}%
8971               {\@argdef#1[#2]}}
8972 \long\def\@argdef#1[#2]#3{%
8973   \@yargdef#1\@ne{#2}{#3}}
8974 \long\def\@xargdef#1[#2][#3]#4{%
8975   \expandafter\def\expandafter#1\expandafter{%
8976     \expandafter\@protected@testopt\expandafter #1%
8977     \csname\string#1\expandafter\endcsname{#3}}%
8978   \expandafter\@yargdef \csname\string#1\endcsname
8979   \tw@{#2}{#4}}
8980 \long\def\@yargdef#1#2#3{%
8981   \@tempcnta#3\relax
8982   \advance \@tempcnta \@ne
8983   \let\@hash@\relax
8984   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8985   \@tempcntb #2%
8986   \@whilenum \@tempcntb <\@tempcnta
8987   \do{%
8988     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8989     \advance\@tempcntb \@ne}%
8990   \let\@hash@###%
8991   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8992 \def\providecommand{\@star@or@long\provide@command}
8993 \def\provide@command#1{%
8994   \begingroup
8995     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8996   \endgroup
8997   \expandafter\ifundefined\@gtempa
8998     {\def\reserved@a{\new@command#1}}%
8999     {\let\reserved@a\relax
9000     \def\reserved@a{\new@command\reserved@a}}%
9001   \reserved@a}%

9002 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9003 \def\declare@robustcommand#1{%
9004   \edef\reserved@a{\string#1}%
9005   \def\reserved@b{#1}%
9006   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9007   \edef#1{%
9008     \ifx\reserved@a\reserved@b
9009       \noexpand\x@protect
9010       \noexpand#1%
9011     \fi
9012     \noexpand\protect
9013     \expandafter\noexpand\csname
9014       \expandafter\@gobble\string#1 \endcsname
9015   }%
9016   \expandafter\new@command\csname
9017     \expandafter\@gobble\string#1 \endcsname
9018 }
9019 \def\x@protect#1{%
9020   \ifx\protect\@typeset@protect\else
9021     \@x@protect#1%
9022   \fi
9023 }
9024 \catcode`\&=\z@ % Trick to hide conditionals

```

```
9025 \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`.

```
9026 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9027 \catcode`\&=4
9028 \ifx\in@\@undefined
9029 \def\in@#1#2{%
9030 \def\in@@##1#1##2##3\in@@{%
9031 \ifx\in@@##2\in@false\else\in@true\fi}%
9032 \in@@##2#1\in@\in@@}
9033 \else
9034 \let\bbl@tempa\@empty
9035 \fi
9036 \bbl@tempa
```

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

```
9037 \def\@ifpackagewith#1#2#3#4{#3}
```

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

```
9038 \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 $\text{\LaTeX 2}_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```
9039 \ifx\@tempcnta\@undefined
9040 \csname newcount\endcsname\@tempcnta\relax
9041 \fi
9042 \ifx\@tempcntb\@undefined
9043 \csname newcount\endcsname\@tempcntb\relax
9044 \fi
```

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

```
9045 \ifx\bye\@undefined
9046 \advance\count10 by -2\relax
9047 \fi
9048 \ifx\@ifnextchar\@undefined
9049 \def\@ifnextchar#1#2#3{%
9050 \let\reserved@d=#1%
9051 \def\reserved@a{#2}\def\reserved@b{#3}%
9052 \futurelet\@let@token\@ifnch}
9053 \def\@ifnch{%
9054 \ifx\@let@token\@sptoken
9055 \let\reserved@c\@xifnch
9056 \else
9057 \ifx\@let@token\reserved@d
9058 \let\reserved@c\reserved@a
9059 \else
9060 \let\reserved@c\reserved@b
9061 \fi
9062 \fi
9063 \reserved@c}
9064 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
9065 \def\{\@xifnch} \expandafter\def\{\futurelet\@let@token\@ifnch}
9066 \fi
9067 \def\@testopt#1#2{%
9068 \@ifnextchar[{\#1}{\#1[#2]}}
```

```

9069 \def\@protected@testopt#1{%
9070   \ifx\protect\@typeset@protect
9071     \expandafter\@testopt
9072   \else
9073     \x@protect#1%
9074   \fi}
9075 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9076   #2\relax}\fi}
9077 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9078   \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9079 \def\DeclareTextCommand{%
9080   \@dec@text@cmd\providecommand
9081 }
9082 \def\ProvideTextCommand{%
9083   \@dec@text@cmd\providecommand
9084 }
9085 \def\DeclareTextSymbol#1#2#3{%
9086   \@dec@text@cmd\chardef#1{#2}#3\relax
9087 }
9088 \def\@dec@text@cmd#1#2#3{%
9089   \expandafter\def\expandafter#2%
9090     \expandafter{%
9091       \csname#3-cmd\expandafter\endcsname
9092       \expandafter#2%
9093       \csname#3\string#2\endcsname
9094     }%
9095 %   \let\@ifdefinable\@rc@ifdefinable
9096   \expandafter#1\csname#3\string#2\endcsname
9097 }
9098 \def\@current@cmd#1{%
9099   \ifx\protect\@typeset@protect\else
9100     \noexpand#1\expandafter\@gobble
9101   \fi
9102 }
9103 \def\@changed@cmd#1#2{%
9104   \ifx\protect\@typeset@protect
9105     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9106       \expandafter\ifx\csname ?\string#1\endcsname\relax
9107         \expandafter\def\csname ?\string#1\endcsname{%
9108           \@changed@x@err{#1}%
9109         }%
9110       \fi
9111       \global\expandafter\let
9112         \csname\cf@encoding \string#1\expandafter\endcsname
9113         \csname ?\string#1\endcsname
9114       \fi
9115       \csname\cf@encoding\string#1%
9116         \expandafter\endcsname
9117     \else
9118       \noexpand#1%
9119     \fi
9120 }
9121 \def\@changed@x@err#1{%
9122   \errhelp{Your command will be ignored, type <return> to proceed}%
9123   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9124 \def\DeclareTextCommandDefault#1{%
9125   \DeclareTextCommand#1%
9126 }
9127 \def\ProvideTextCommandDefault#1{%

```

```

9128 \ProvideTextCommand#1?%
9129 }
9130 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9131 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9132 \def\DeclareTextAccent#1#2#3{%
9133 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
9134 }
9135 \def\DeclareTextCompositeCommand#1#2#3#4{%
9136 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9137 \edef\reserved@b{\string##1}%
9138 \edef\reserved@c{%
9139 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9140 \ifx\reserved@b\reserved@c
9141 \expandafter\expandafter\expandafter\ifx
9142 \expandafter\@car\reserved@a\relax\relax\@nil
9143 \@text@composite
9144 \else
9145 \edef\reserved@b##1{%
9146 \def\expandafter\noexpand
9147 \csname#2\string#1\endcsname###1{%
9148 \noexpand\@text@composite
9149 \expandafter\noexpand\csname#2\string#1\endcsname
9150 ###1\noexpand\@empty\noexpand\@text@composite
9151 {##1}%
9152 }%
9153 }%
9154 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9155 \fi
9156 \expandafter\def\csname\expandafter\string\csname
9157 #2\endcsname\string#1-\string#3\endcsname{#4}
9158 \else
9159 \errhelp{Your command will be ignored, type <return> to proceed}%
9160 \errmessage{\string\DeclareTextCompositeCommand\space used on
9161 inappropriate command \protect#1}
9162 \fi
9163 }
9164 \def\@text@composite#1#2#3\@text@composite{%
9165 \expandafter\@text@composite@x
9166 \csname\string#1-\string#2\endcsname
9167 }
9168 \def\@text@composite@x#1#2{%
9169 \ifx#1\relax
9170 #2%
9171 \else
9172 #1%
9173 \fi
9174 }
9175 %
9176 \def\@strip@args#1:#2-#3\@strip@args{#2}
9177 \def\DeclareTextComposite#1#2#3#4{%
9178 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9179 \bgroup
9180 \lccode`\@=#4%
9181 \lowercase{%
9182 \egroup
9183 \reserved@a @%
9184 }%
9185 }
9186 %
9187 \def\UseTextSymbol#1#2{#2}
9188 \def\UseTextAccent#1#2#3{}
9189 \def\@use@text@encoding#1{}
9190 \def\DeclareTextSymbolDefault#1#2{%

```

```

9191 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9192 }
9193 \def\DeclareTextAccentDefault#1#2{%
9194 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9195 }
9196 \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.

```

9197 \DeclareTextAccent{"}{OT1}{127}
9198 \DeclareTextAccent{'}{OT1}{19}
9199 \DeclareTextAccent{^}{OT1}{94}
9200 \DeclareTextAccent{`}{OT1}{18}
9201 \DeclareTextAccent{~}{OT1}{126}

```

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

```

9202 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9203 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
9204 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
9205 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
9206 \DeclareTextSymbol{\i}{OT1}{16}
9207 \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`.

```

9208 \ifx\scriptsize\undefined
9209 \let\scriptsize\sevenrm
9210 \fi

```

And a few more “dummy” definitions.

```

9211 \def\language{english}%
9212 \let\bbl@opt@shorthands\@nnil
9213 \def\bbl@ifshorthand#1#2#3{#2}%
9214 \let\bbl@language@opts\@empty
9215 \let\bbl@provide@locale\relax
9216 \ifx\babeloptionstrings\undefined
9217 \let\bbl@opt@strings\@nnil
9218 \else
9219 \let\bbl@opt@strings\babeloptionstrings
9220 \fi
9221 \def\BabelStringsDefault{generic}
9222 \def\bbl@tempa{normal}
9223 \ifx\babeloptionmath\bbl@tempa
9224 \def\bbl@mathnormal{\noexpand\textormath}
9225 \fi
9226 \def\AfterBabelLanguage#1#2{}
9227 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9228 \let\bbl@afterlang\relax
9229 \def\bbl@opt@safe{BR}
9230 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
9231 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9232 \expandafter\newif\csname ifbbl@single\endcsname
9233 \chardef\bbl@bidimode\z@
9234 <</Emulate LaTeX>>

```

A proxy file:

```

9235 <*\plain>
9236 \input babel.def
9237 </\plain>

```

15. Acknowledgements

In the initial stages of the development of `babel`, Bernd Raichle provided many helpful suggestions and Michel Goossens supplied contributions for many languages. Ideas from Nico Poppelier, Piet van

Oostrum and many others have been used. Paul Wackers and Werenfried Spit helped find and repair bugs.

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

Barbara Beeton has helped in improving the manual.

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

References

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