

Babel

Code

Version 24.15.72974
2024/12/29

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	7
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)	12
4	babel.sty and babel.def (common)	13
4.1	Selecting the language	15
4.2	Errors	22
4.3	More on selection	23
4.4	Short tags	25
4.5	Compatibility with language.def	25
4.6	Hooks	26
4.7	Setting up language files	26
4.8	Shorthands	28
4.9	Language attributes	37
4.10	Support for saving and redefining macros	39
4.11	French spacing	40
4.12	Hyphens	41
4.13	Multiencoding strings	43
4.14	Tailor captions	47
4.15	Making glyphs available	48
4.15.1	Quotation marks	48
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	53
4.18	Creating and modifying languages	53
4.19	Main loop in ‘provide’	61
4.20	Processing keys in ini	64
4.21	French spacing (again)	69
4.22	Handle language system	71
4.23	Numerals	72
4.24	Casing	73
4.25	Getting info	74
4.26	BCP 47 related commands	75
5	Adjusting the Babel behavior	76
5.1	Cross referencing macros	78
5.2	Layout	81
5.3	Marks	81
5.4	Other packages	82
5.4.1	ifthen	82
5.4.2	varioref	83
5.4.3	hhline	83
5.5	Encoding and fonts	84
5.6	Basic bidi support	86
5.7	Local Language Configuration	89
5.8	Language options	89

6	The kernel of Babel	93
7	Error messages	93
8	Loading hyphenation patterns	96
9	luatex + xetex: common stuff	100
10	Hooks for XeTeX and LuaTeX	104
10.1	XeTeX	104
10.2	Support for interchar	106
10.3	Layout	108
10.4	8-bit TeX	109
10.5	LuaTeX	110
10.6	Southeast Asian scripts	117
10.7	CJK line breaking	118
10.8	Arabic justification	120
10.9	Common stuff	124
10.10	Automatic fonts and ids switching	124
10.11	Bidi	131
10.12	Layout	133
10.13	Lua: transforms	143
10.14	Lua: Auto bidi with basic and basic-r	152
11	Data for CJK	164
12	The ‘nil’ language	164
13	Calendars	165
13.1	Islamic	165
13.2	Hebrew	167
13.3	Persian	171
13.4	Coptic and Ethiopic	171
13.5	Buddhist	172
14	Support for Plain T_EX (plain.def)	173
14.1	Not renaming hyphen.tex	173
14.2	Emulating some L ^A T _E X features	174
14.3	General tools	175
14.4	Encoding related macros	178
15	Acknowledgements	181

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=24.15.72974>>
2 <<date=2024/12/29>>
```

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

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

```

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

```

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

```

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

```

\bbl@afterelse

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

```

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

```

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

```

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

```

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

```

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

```

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

\bbl@ifunset To check if a macro is defined, we create a new macro, which does the same as `\ifundefined`. However, in an ϵ -tex engine, it is based on `\ifcsname`, which is more efficient, and does not waste memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

```

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

```

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

```

76 \def\bbl@ifblank#1{%
77   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
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}}

```

\bbl@replace Returns implicitly `\toks@` with the modified string.

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3

```

```

102 \toks@{}%
103 \def\bbl@replace@aux##1#2##2#2{%
104   \ifx\bbl@nil##2%
105     \toks@\expandafter{\the\toks@##1}%
106   \else
107     \toks@\expandafter{\the\toks@##1#3}%
108     \bbl@afterfi
109     \bbl@replace@aux##2#2%
110   \fi}%
111 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112 \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).

```

113 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
114   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax}%
115   \def\bbl@tempa{#1}%
116   \def\bbl@tempb{#2}%
117   \def\bbl@tempe{#3}}
118 \def\bbl@sreplace#1#2#3{%
119   \begingroup
120     \expandafter\bbl@parsedef\meaning#1\relax
121     \def\bbl@tempc{#2}%
122     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
123     \def\bbl@tempd{#3}%
124     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
125     \bbl@xin{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
126     \ifin@
127       \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
128       \def\bbl@tempc{Expanded an executed below as 'uplevel'
129         \\makeatletter % "internal" macros with @ are assumed
130         \\scantokens{
131           \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132         \catcode64=\the\catcode64\relax}% Restore @
133     \else
134       \let\bbl@tempc\empty % Not \relax
135     \fi
136     \bbl@exp{For the 'uplevel' assignments
137   \endgroup
138   \bbl@tempc}} % empty or expand to set #1 with changes
139 \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 pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

140 \def\bbl@ifsamestring#1#2{%
141   \begingroup
142     \protected@edef\bbl@tempb{#1}%
143     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
144     \protected@edef\bbl@tempc{#2}%
145     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
146     \ifx\bbl@tempb\bbl@tempc
147       \aftergroup\@firstoftwo
148     \else
149       \aftergroup\@secondoftwo
150     \fi
151   \endgroup}
152 \chardef\bbl@engine=
153 \ifx\directlua\undefined
154   \ifx\XeTeXinputencoding\undefined

```

```

155     \z@
156   \else
157     \tw@
158   \fi
159 \else
160   \@ne
161 \fi

```

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

```

162 \def\bbl@bsphack{%
163   \ifhmode
164     \hskip\z@skip
165   \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166   \else
167     \let\bbl@esphack\@empty
168   \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`.

```

169 \def\bbl@cased{%
170   \ifx\oe\OE
171     \expandafter\in@\expandafter
172       {\expandafter\OE\expandafter}\expandafter{\oe}%
173   \ifin@
174     \bbl@afterelse\expandafter\MakeUppercase
175   \else
176     \bbl@afterfi\expandafter\MakeLowercase
177   \fi
178 \else
179   \expandafter\@firstofone
180 \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`).

```

181 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182   \toks@{\expandafter\expandafter\expandafter}%
183   \csname extras\language\endcsname}%
184   \bbl@exp{\in@{#1}{\the\toks@}}%
185   \ifin@\else
186     \@temptokena{#2}%
187     \edef\bbl@tempc{\the\@temptokena\the\toks@}%
188     \toks@\expandafter{\bbl@tempc#3}%
189     \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
190   \fi}
191 <</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 .

```

192 <<*Make sure ProvidesFile is defined>> ≡
193 \ifx\ProvidesFile\@undefined
194   \def\ProvidesFile#1[#2 #3 #4]{%
195     \wlog{File: #1 #4 #3 <#2>}%
196     \let\ProvidesFile\@undefined}
197 \fi
198 <</Make sure ProvidesFile is defined>>

```

3.1. A few core definitions

Language Just for compatibility, for not to touch `hyphen.cfg`.

```

199 <<*Define core switching macros>> ≡
200 \ifx\language\@undefined
201   \csname newcount\endcsname\language
202 \fi
203 <</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.

```
204 <<*Define core switching macros>> ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 <</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.

```
208 <*package>
209 \NeedsTeXFormat{LaTeX2e}
210 \ProvidesPackage{babel}%
211 [ <@date@> v<@version@> %%NB%%
212 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).

```
213 \@ifpackagewith{babel}{debug}
214 {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
215 \let\bbl@debug@firstofone
216 \ifx\directlua\undefined\else
217 \directlua{
218   Babel = Babel or {}
219   Babel.debug = true }%
220 \input{babel-debug.tex}%
221 \fi}
222 {\providecommand\bbl@trace[1]{}%
223 \let\bbl@debug@gobble
224 \ifx\directlua\undefined\else
225 \directlua{
226   Babel = Babel or {}
227   Babel.debug = false }%
228 \fi}
```

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

```
229 \def\bbl@error#1{% Implicit #2#3#4
230 \begingroup
231 \catcode`\=0 \catcode`\==12 \catcode`\`=12
232 \input errbabel.def
233 \endgroup
234 \bbl@error{#1}}
235 \def\bbl@warning#1{%
236 \begingroup
237 \def\{\{MessageBreak}%
238 \PackageWarning{babel}{#1}%
239 \endgroup}
240 \def\bbl@infowarn#1{%
241 \begingroup
242 \def\{\{MessageBreak}%
243 \PackageNote{babel}{#1}%
```

```

244 \endgroup}
245 \def\bbl@info#1{%
246 \begingroup
247 \def\{\MessageBreak}%
248 \PackageInfo{babel}{#1}%
249 \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.

```

250 <@Basic macros>
251 \ifpackagewith{babel}{silent}
252 {\let\bbl@info@gobble
253 \let\bbl@infowarn@gobble
254 \let\bbl@warning@gobble}
255 {}
256 %
257 \def\AfterBabelLanguage#1{%
258 \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.

```

259 \ifx\bbl@languages@undefined\else
260 \begingroup
261 \catcode\^^I=12
262 \ifpackagewith{babel}{showlanguages}{%
263 \begingroup
264 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
265 \wlog{<*languages>}%
266 \bbl@languages
267 \wlog{</languages>}%
268 \endgroup}{%
269 \endgroup
270 \def\bbl@elt#1#2#3#4{%
271 \ifnum#2=\z@
272 \gdef\bbl@nulllanguage{#1}%
273 \def\bbl@elt##1##2##3##4{%
274 \fi}%
275 \bbl@languages
276 \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.

```

277 \bbl@trace{Defining option 'base'}
278 \ifpackagewith{babel}{base}{%
279 \let\bbl@onlyswitch@empty
280 \let\bbl@provide@locale@relax
281 \input babel.def
282 \let\bbl@onlyswitch@undefined
283 \ifx\directlua@undefined
284 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
285 \else
286 \input luababel.def
287 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
288 \fi
289 \DeclareOption{base}{}%
290 \DeclareOption{showlanguages}{}%
291 \ProcessOptions

```

```

292 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
293 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
294 \global\let\@ifl@ter@@\@ifl@ter
295 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
296 \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`.

```

297 \bbl@trace{key=value and another general options}
298 \bbl@csarg\let\tempa\expandafter\csname opt@babel.sty\endcsname
299 \def\bbl@tempb#1.#2{% Remove trailing dot
300   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
301 \def\bbl@tempe#1=#2\@@{%
302   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
303 \def\bbl@tempd#1.#2\@nnil{%^A TODO. Refactor lists?
304   \ifx\@empty#2%
305     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
306   \else
307     \in@{,provide=}{, #1}%
308     \ifin@
309       \edef\bbl@tempc{%
310         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
311     \else
312       \in@{$modifiers$}{$#1$}%^A TODO. Allow spaces.
313       \ifin@
314         \bbl@tempe#2\@@
315       \else
316         \in@{=}{#1}%
317         \ifin@
318           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
319         \else
320           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
321           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
322         \fi
323       \fi
324     \fi
325   \fi}
326 \let\bbl@tempc\@empty
327 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
328 \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.

```

329 \DeclareOption{KeepShorthandsActive}{}
330 \DeclareOption{activeacute}{}
331 \DeclareOption{activegrave}{}
332 \DeclareOption{debug}{}
333 \DeclareOption{noconfigs}{}
334 \DeclareOption{showlanguages}{}
335 \DeclareOption{silent}{}
336 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
337 \chardef\bbl@iniflag\z@
338 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
339 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
340 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
341 % Don't use. Experimental. TODO.
342 \newif\ifbbl@single
343 \DeclareOption{selectors=off}{\bbl@singletrue}
344 <@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.

```

345 \let\bbl@opt@shorthands\@nnil
346 \let\bbl@opt@config\@nnil
347 \let\bbl@opt@main\@nnil
348 \let\bbl@opt@headfoot\@nnil
349 \let\bbl@opt@layout\@nnil
350 \let\bbl@opt@provide\@nnil

```

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

```

351 \def\bbl@tempa#1=#2\bbl@tempa{%
352   \bbl@csarg\ifx{opt@#1}\@nnil
353   \bbl@csarg\edef{opt@#1}{#2}%
354   \else
355   \bbl@error{bad-package-option}{#1}{#2}{}%
356   \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.

```

357 \let\bbl@language@opts\@empty
358 \DeclareOption*{%
359   \bbl@xin@{\string=}{\CurrentOption}%
360   \ifin@
361   \expandafter\bbl@tempa\CurrentOption\bbl@tempa
362   \else
363   \bbl@add@list\bbl@language@opts{\CurrentOption}%
364   \fi}

```

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

```

365 \ProcessOptions*

```

3.5. Post-process some options

```

366 \ifx\bbl@opt@provide\@nnil
367   \let\bbl@opt@provide\@empty % %%% MOVE above
368 \else
369   \chardef\bbl@iniflag\@ne
370   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
371     \in@{,provide,}{, #1,}%
372     \ifin@
373     \def\bbl@opt@provide{#2}%
374     \fi}
375 \fi

```

If there is no shorthands= $\langle chars \rangle$, 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=...

```

376 \bbl@trace{Conditional loading of shorthands}
377 \def\bbl@sh@string#1{%
378   \ifx#1\@empty\else
379   \ifx#1t\string~%
380   \else\ifx#1c\string,%
381   \else\string#1%
382   \fi\fi
383   \expandafter\bbl@sh@string
384   \fi}
385 \ifx\bbl@opt@shorthands\@nnil
386   \def\bbl@ifshorthand#1#2#3{#2}%
387 \else\ifx\bbl@opt@shorthands\@empty
388   \def\bbl@ifshorthand#1#2#3{#3}%

```

```
389 \else
```

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

```
390 \def\bbl@ifshorthand#1{%
391   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
392   \ifin@
393     \expandafter\@firstoftwo
394   \else
395     \expandafter\@secondoftwo
396   \fi}
```

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

```
397 \edef\bbl@opt@shorthands{%
398   \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.

```
399 \bbl@ifshorthand{'}%
400   {\PassOptionsToPackage{activeacute}{babel}}{}
401 \bbl@ifshorthand{`}%
402   {\PassOptionsToPackage{activegrave}{babel}}{}
403 \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.

```
404 \ifx\bbl@opt@headfoot\@nnil\else
405   \g@addto@macro\@resetactivechars{%
406     \set@typeset@protect
407     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
408     \let\protect\noexpand}
409 \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.

```
410 \ifx\bbl@opt@safe\@undefined
411   \def\bbl@opt@safe{BR}
412   % \let\bbl@opt@safe\@empty % Pending of \cite
413 \fi
```

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

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

```
414 \bbl@trace{Defining IfBabelLayout}
415 \ifx\bbl@opt@layout\@nnil
416   \newcommand\IfBabelLayout[3]{#3}%
417 \else
418   \bbl@exp{\bbl@forkv{\@nameuse{\raw@opt@babel.sty}}}{%
419     \in@{, layout,}{, #1,}%
420     \ifin@
421       \def\bbl@opt@layout{#2}%
422       \bbl@replace\bbl@opt@layout{ }{.}%
423     \fi}
424   \newcommand\IfBabelLayout[1]{%
425     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
426     \ifin@
427       \expandafter\@firstoftwo
428     \else
429       \expandafter\@secondoftwo
430     \fi}
431 \fi
432 \</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.

```
433 < *core >
434 \ifx\ldf@quit\@undefined\else
435 \endinput\fi % Same line!
436 <@Make sure ProvidesFile is defined@>
437 \ProvidesFile{babel.def}[<@date@> v<@version@> Babel common definitions]
438 \ifx\AtBeginDocument\@undefined %^^A TODO. change test.
439 <@Emulate LaTeX@>
440 \fi
441 <@Basic macros@>
442 < /core >
```

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

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

```
443 < *package | core >
444 \def\bbl@version{<@version@>}
445 \def\bbl@date{<@date@>}
446 <@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.

```
447 \def\adddialect#1#2{%
448   \global\chardef#1#2\relax
449   \bbl@usehooks{adddialect}{#{1}{#2}}%
450   \begingroup
451     \count@#1\relax
452     \def\bbl@elt##1##2###3###4{%
453       \ifnum\count@=##2\relax
454         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
455         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
456               set to \expandafter\string\csname l@##1\endcsname\\%
457               (\string\language\the\count@). Reported}%
458         \def\bbl@elt####1####2####3####4{%
459           \fi}%
460       \bbl@cs{languages}%
461       \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.

```
462 \def\bbl@fixname#1{%
463   \begingroup
464     \def\bbl@tempe{l@}%
465     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
466     \bbl@tempd
467     {\lowercase\expandafter{\bbl@tempd}%
468      {\uppercase\expandafter{\bbl@tempd}%
469       \@empty
470       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
471        {\uppercase\expandafter{\bbl@tempd}}}%
472       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
473        {\lowercase\expandafter{\bbl@tempd}}}%
474       \@empty
475       \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
476     \bbl@tempd
477     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}
478 \def\bbl@iflanguage#1{%
```

```
479 \ifundefined{#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`.

```
480 \def\bbl@bcpcase#1#2#3#4\@#5{%
481   \ifx\@empty#3%
482     \uppercase{\def#5{#1#2}}%
483   \else
484     \uppercase{\def#5{#1}}%
485     \lowercase{\edef#5{#5#2#3#4}}%
486   \fi}
487 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
488   \let\bbl@bcp\relax
489   \lowercase{\def\bbl@tempa{#1}}%
490   \ifx\@empty#2%
491     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
492   \else\ifx\@empty#3%
493     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb}
494     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
495       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
496       {}%
497     \ifx\bbl@bcp\relax
498       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
499     \fi
500   \else
501     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb}
502     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc}
503     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
504       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
505       {}%
506     \ifx\bbl@bcp\relax
507       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
508       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
509       {}%
510     \fi
511     \ifx\bbl@bcp\relax
512       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
513       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
514       {}%
515     \fi
516     \ifx\bbl@bcp\relax
517       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
518     \fi
519   \fi\fi}
520 \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.

```
521 \def\iflanguage#1{%
522   \bbl@iflanguage{#1}{%
523     \ifnum\csname l@#1\endcsname=\language
524       \expandafter\@firstoftwo
525     \else
526       \expandafter\@secondoftwo
527     \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.

```
528 \let\bbl@select@type\z@
529 \edef\selectlanguage{%
530   \noexpand\protect
531   \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.

```
532 \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.

```
533 \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.

```
534 \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:

```
535 \def\bbl@push@language{%
536   \ifx\language\@undefined\else
537     \ifx\currentgrouplevel\@undefined
538       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
539     \else
540       \ifnum\currentgrouplevel=\z@
541         \xdef\bbl@language@stack{\language+}%
542       \else
543         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
544       \fi
545     \fi
546   \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.

```
547 \def\bbl@pop@lang#1+#2\@{%
548   \edef\language{#1}%
549   \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).

```
550 \let\bbl@ifrestoring\@secondoftwo
551 \def\bbl@pop@language{%
552   \expandafter\bbl@pop@lang\bbl@language@stack\@
553   \let\bbl@ifrestoring\@firstoftwo
554   \expandafter\bbl@set@language\expandafter{\language}%
555   \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).

```
556 \chardef\localeid\z@
557 \def\bbl@id@last{0} % No real need for a new counter
558 \def\bbl@id@assign{%
559   \bbl@ifunset\bbl@id@\language}%
560   {\count\bbl@id@last\relax
561    \advance\count@one
562    \global\bbl@csarg\chardef{id@\language}\count@
563    \edef\bbl@id@last{\the\count@}%
564    \ifcase\bbl@engine\or
565      \directlua{
566        Babel.locale_props[\bbl@id@last] = {}
567        Babel.locale_props[\bbl@id@last].name = '\language'
568        Babel.locale_props[\bbl@id@last].vars = {}
569      }%
570    \fi}%
571  }%
572  \chardef\localeid\bbl@c{id@}
```

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

```
573 \expandafter\def\csname selectlanguage \endcsname#1{%
574   \ifnum\bbl@hymapsel=\ccclv\let\bbl@hymapsel\tw@\fi
575   \bbl@push@language
576   \aftergroup\bbl@pop@language
577   \bbl@set@language{#1}}
578 \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` or `\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@save@lastskip` 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).

```
579 \def\BabelContentsFiles{toc,lof,lot}
580 \def\bbl@set@language#1{% from selectlanguage, pop@
581   % The old buggy way. Preserved for compatibility, but simplified
582   \edef\language{\expandafter\string#1\empty}%
583   \select@language{\language}%
```

```

584 % write to auxs
585 \expandafter\ifx\csname date\language\endcsname\relax\else
586   \if@files
587     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
588       \bbl@savelastskip
589       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
590       \bbl@restorelastskip
591     \fi
592     \bbl@usehooks{write}}}%
593   \fi
594 \fi}
595 %
596 \let\bbl@restorelastskip\relax
597 \let\bbl@savelastskip\relax
598 %
599 \def\select@language#1{% from set@, babel@aux, babel@toc
600   \ifx\bbl@selectorname\empty
601     \def\bbl@selectorname{select}%
602   \fi
603   % set hymap
604   \ifnum\bbl@hymapset=\@ccclv\chardef\bbl@hymapset4\relax\fi
605   % set name (when coming from babel@aux)
606   \edef\language{#1}%
607   \bbl@fixname\language
608   % define \locale when coming from set@, with a trick
609   \ifx\scantokens\undefined
610     \def\locale{??}%
611   \else
612     \bbl@exp{\scantokens{\def\locale{\language}\noexpand}\relax}%
613   \fi
614   %^^A TODO. name@map must be here?
615   \bbl@provide@locale
616   \bbl@iflanguage\language{%
617     \let\bbl@select@type\z@
618     \expandafter\bbl@switch\expandafter{\language}}
619 \def\babel@aux#1#2{%
620   \select@language{#1}%
621   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
622     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}%^^A TODO - plain?
623 \def\babel@toc#1#2{%
624   \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`.

```

625 \newif\ifbbl@usedategroup
626 \let\bbl@savextras\empty
627 \def\bbl@switch#1{% from select@, foreign@
628   % make sure there is info for the language if so requested
629   \bbl@ensureinfo{#1}%
630   % restore
631   \originalTeX

```

```

632 \expandafter\def\expandafter\originalTeX\expandafter{%
633   \csname noextras#1\endcsname
634   \let\originalTeX\@empty
635   \babel@beginsave}%
636 \bbl@usehooks{afterreset}{}%
637 \languageshorthands{none}%
638 % set the locale id
639 \bbl@id@assign
640 % switch captions, date
641 \bbl@bsphack
642 \ifcase\bbl@select@type
643   \csname captions#1\endcsname\relax
644   \csname date#1\endcsname\relax
645 \else
646   \bbl@xin@{,captions,}{,\bbl@select@opts,}%
647   \ifin@
648     \csname captions#1\endcsname\relax
649   \fi
650   \bbl@xin@{,date,}{,\bbl@select@opts,}%
651   \ifin@ % if \foreign... within \<language>date
652     \csname date#1\endcsname\relax
653   \fi
654 \fi
655 \bbl@esphack
656 % switch extras
657 \csname bbl@preextras@#1\endcsname
658 \bbl@usehooks{beforeextras}{}%
659 \csname extras#1\endcsname\relax
660 \bbl@usehooks{afterextras}{}%
661 % > babel-ensure
662 % > babel-sh-<short>
663 % > babel-bidi
664 % > babel-fontspec
665 \let\bbl@savextras\@empty
666 % hyphenation - case mapping
667 \ifcase\bbl@opt@hyphenmap\or
668   \def\BabelLower##1##2{\lccode##1=##2\relax}%
669   \ifnum\bbl@hymap>4\else
670     \csname\language @bbl@hyphenmap\endcsname
671   \fi
672   \chardef\bbl@opt@hyphenmap\z@
673 \else
674   \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
675     \csname\language @bbl@hyphenmap\endcsname
676   \fi
677 \fi
678 \let\bbl@hymap\@cclv
679 % hyphenation - select rules
680 \ifnum\csname l@language\endcsname=\l@unhyphenated
681   \edef\bbl@tempa{u}%
682 \else
683   \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
684 \fi
685 % linebreaking - handle u, e, k (v in the future)
686 \bbl@xin@{/u}{/\bbl@tempa}%
687 \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
688 \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
689 \ifin@ \else \bbl@xin@{/p}{/\bbl@tempa} \fi % padding (e.g., Tibetan)
690 \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
691 % hyphenation - save mins
692 \babel@savevariable\lefthyphenmin
693 \babel@savevariable\righthyphenmin
694 \ifnum\bbl@engine=\@ne

```

```

695 \babel@savevariable\hyphenationmin
696 \fi
697 \ifin@
698 % unhyphenated/kashida/elongated/padding = allow stretching
699 \language\l@unhyphenated
700 \babel@savevariable\emergencystretch
701 \emergencystretch\maxdimen
702 \babel@savevariable\hbadness
703 \hbadness\@M
704 \else
705 % other = select patterns
706 \bbl@patterns{#1}%
707 \fi
708 % hyphenation - set mins
709 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
710 \set@hyphenmins\tw@\thr@@\relax
711 \@nameuse{bbl@hyphenmins@}%
712 \else
713 \expandafter\expandafter\expandafter\set@hyphenmins
714 \csname #1hyphenmins\endcsname\relax
715 \fi
716 \@nameuse{bbl@hyphenmins@}%
717 \@nameuse{bbl@hyphenmins@\language\language}%
718 \@nameuse{bbl@hyphenatmin@}%
719 \@nameuse{bbl@hyphenatmin@\language\language}%
720 \let\bbl@selectorname\empty}

```

otherlanguage It can be used as an alternative to using the `\selectlanguage` declarative command. The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

721 \long\def\otherlanguage#1{%
722 \def\bbl@selectorname{other}%
723 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
724 \csname selectlanguage \endcsname{#1}%
725 \ignorespaces}

```

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

```

726 \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`.

```

727 \expandafter\def\csname otherlanguage*\endcsname{%
728 \ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
729 \def\bbl@otherlanguage@s[#1]#2{%
730 \def\bbl@selectorname{other*}%
731 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
732 \def\bbl@select@opts{#1}%
733 \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”.

```

734 \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.

```

735 \providecommand\bbl@beforeforeign{}
736 \edef\foreignlanguage{%
737   \noexpand\protect
738   \expandafter\noexpand\csname foreignlanguage \endcsname}
739 \expandafter\def\csname foreignlanguage \endcsname{%
740   \@ifstar\bbl@foreign@s\bbl@foreign@x}
741 \providecommand\bbl@foreign@x[3][{}]{%
742   \beginngroup
743     \def\bbl@select@name{foreign}%
744     \def\bbl@select@opts{#1}%
745     \let\BabelText\@firstofone
746     \bbl@beforeforeign
747     \foreign@language{#2}%
748     \bbl@usehooks{foreign}{}%
749     \BabelText{#3}% Now in horizontal mode!
750   \endgroup}
751 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@@par
752   \beginngroup
753     {\par}%
754     \def\bbl@select@name{foreign*}%
755     \let\bbl@select@opts\@empty
756     \let\BabelText\@firstofone
757     \foreign@language{#1}%
758     \bbl@usehooks{foreign*}{}%
759     \bbl@dirparastext
760     \BabelText{#2}% Still in vertical mode!
761     {\par}%
762   \endgroup}
763 \providecommand\BabelWrapText[1]{%
764   \def\bbl@tempa{\def\BabelText###1}%
765   \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`.

```

766 \def\foreign@language#1{%
767   % set name
768   \edef\language#1%
769   \ifbbl@usedatagroup
770     \bbl@add\bbl@select@opts{,date,}%
771     \bbl@usedatagroupfalse
772   \fi
773   \bbl@fixname\language
774   \let\localname\language
775   % TODO. name@map here?
776   \bbl@provide@locale
777   \bbl@iflanguage\language%
778     \let\bbl@select@type\@ne
```

```
779 \expandafter\bb1@switch\expandafter{\language\name}}
```

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

```
780 \def\IfBabelSelectorTF#1{%
781 \bb1@xin{\bb1@selectorname,}{,\zap@space#1 \empty},}%
782 \ifin@
783 \expandafter\@firstoftwo
784 \else
785 \expandafter\@secondoftwo
786 \fi}
```

\bb1@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). \bb1@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.

```
787 \let\bb1@hyphlist\empty
788 \let\bb1@hyphenation@\relax
789 \let\bb1@pttnlist\empty
790 \let\bb1@patterns@\relax
791 \let\bb1@hymapsel=\@cclv
792 \def\bb1@patterns#1{%
793 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
794 \csname l@#1\endcsname
795 \edef\bb1@tempa{#1}%
796 \else
797 \csname l@#1:\f@encoding\endcsname
798 \edef\bb1@tempa{#1:\f@encoding}%
799 \fi
800 \@expandtwoargs\bb1@usehooks{patterns}{\bb1@tempa}}%
801 % > luatex
802 \@ifundefined{bb1@hyphenation@}{\relax!
803 \begin{group}
804 \bb1@xin{\number\language,}{,\bb1@hyphlist}%
805 \ifin@\else
806 \@expandtwoargs\bb1@usehooks{hyphenation}{\bb1@tempa}}%
807 \hyphenation%
808 \bb1@hyphenation@
809 \@ifundefined{bb1@hyphenation@#1}%
810 \empty
811 {\space\csname bb1@hyphenation@#1\endcsname}}%
812 \xdef\bb1@hyphlist{\bb1@hyphlist\number\language,}%
813 \fi
814 \end{group}}
```

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

```
815 \def\hyphenrules#1{%
816 \edef\bb1@tempf{#1}%
817 \bb1@fixname\bb1@tempf
818 \bb1@iflanguage\bb1@tempf{%
819 \expandafter\bb1@patterns\expandafter{\bb1@tempf}%
820 \ifx\language\shorthand\undefined\else
821 \language\shorthand{none}%
822 \fi
823 \expandafter\ifx\csname bb1@tempf hyphenmins\endcsname\relax
824 \set@hyphenmins\tw@\thr@@\relax
825 \else
```

```

826 \expandafter\expandafter\expandafter\set@hyphenmins
827 \csname\bbl@tempf hyphenmins\endcsname\relax
828 \fi}}
829 \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.

```

830 \def\providehyphenmins#1#2{%
831 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
832 \@namedef{#1hyphenmins}{#2}%
833 \fi}

```

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

```

834 \def\set@hyphenmins#1#2{%
835 \lefthyphenmin#1\relax
836 \righthyphenmin#2\relax}

```

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

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

```

837 \ifx\ProvidesFile\@undefined
838 \def\ProvidesLanguage#1[#2 #3 #4]{%
839 \wlog{Language: #1 #4 #3 <#2>}%
840 }
841 \else
842 \def\ProvidesLanguage#1{%
843 \begingroup
844 \catcode`\ 10 %
845 \@makeother\/%
846 \@ifnextchar[%]
847 {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
848 \def\@provideslanguage#1[#2]{%
849 \wlog{Language: #1 #2}%
850 \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
851 \endgroup}
852 \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`.

```

853 \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.

```

854 \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’:

```

855 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}{}
856 \let\uselocale\setlocale
857 \let\locale\setlocale
858 \let\selectlocale\setlocale
859 \let\textlocale\setlocale
860 \let\textlanguage\setlocale
861 \let\languagegetext\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 L^AT_EX 2_ε, 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.

```

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

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

892 \ifx\bbl@onlyswitch\@empty\endinput\fi

```

4.3. More on selection

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

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

```

893 \bbl@trace{Defining babelensure}
894 \newcommand\babelensure[2][]{%
895   \AddBabelHook{babel-ensure}{afterextras}{%
896     \ifcase\bbl@select@type
897       \bbl@cl{e}%

```



```

898 \fi}%
899 \begingroup
900 \let\bbl@ens@include\@empty
901 \let\bbl@ens@exclude\@empty
902 \def\bbl@ens@fontenc{\relax}%
903 \def\bbl@tempb##1{%
904 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
905 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
906 \def\bbl@tempb##1=##2\@@{\@namedef\bbl@ens@##1}{##2}}%
907 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
908 \def\bbl@tempc{\bbl@ensure}%
909 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
910 \expandafter\bbl@ens@include}%
911 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
912 \expandafter\bbl@ens@exclude}%
913 \toks@\expandafter\bbl@tempc}%
914 \bbl@exp{%
915 \endgroup
916 \def<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
917 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
918 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
919 \ifx##1\@undefined % 3.32 - Don't assume the macro exists
920 \edef##1{\noexpand\bbl@nocaption
921 {\bbl@stripslash##1}{\language\name\bbl@stripslash##1}}}%
922 \fi
923 \ifx##1\@empty\else
924 \in@{##1}{#2}%
925 \ifin@ \else
926 \bbl@ifunset{\bbl@ensure@ \language\name}%
927 {\bbl@exp{%
928 \\\DeclareRobustCommand\<bbl@ensure@ \language\name>[1]{%
929 \\\foreignlanguage{\language\name}%
930 {\ifx\relax#3\else
931 \\\fontencoding{#3}\selectfont
932 \fi
933 #####1}}}%
934 }%
935 \toks@ \expandafter{##1}%
936 \edef##1{%
937 \bbl@csarg\noexpand{ensure@ \language\name}%
938 {\the\toks@}}%
939 \fi
940 \expandafter\bbl@tempb
941 \fi}%
942 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
943 \def\bbl@tempa##1{% elt for include list
944 \ifx##1\@empty\else
945 \bbl@csarg\in@{ensure@ \language\name\expandafter}\expandafter{##1}%
946 \ifin@ \else
947 \bbl@tempb##1\@empty
948 \fi
949 \expandafter\bbl@tempa
950 \fi}%
951 \bbl@tempa#1\@empty}
952 \def\bbl@captionslist{%
953 \prefacename\refname\abstractname\bibname\chaptername\appendixname
954 \contentsname\listfigurename\listtablename\indexname\figurename
955 \tablename\partname\enclname\ccname\headtoname\pagename\seename
956 \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.

```
957 \bbl@trace{Short tags}
958 \newcommand\babeltags[1]{%
959   \edef\bbl@tempa{\zap@space#1 \@empty}%
960   \def\bbl@tempb##1=##2\@{
961     \edef\bbl@tempc{%
962       \noexpand\newcommand
963       \expandafter\noexpand\csname ##1\endcsname{%
964         \noexpand\protect
965         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
966       \noexpand\newcommand
967       \expandafter\noexpand\csname text##1\endcsname{%
968         \noexpand\foreignlanguage{##2}}
969     \bbl@tempc}%
970   \bbl@for\bbl@tempa\bbl@tempa{%
971     \expandafter\bbl@tempb\bbl@tempa\@{}}
```

4.5. Compatibility with language.def

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

```
972 \bbl@trace{Compatibility with language.def}
973 \ifx\directlua\@undefined\else
974   \ifx\bbl@luapatterns\@undefined
975     \input luabel.def
976   \fi
977 \fi
978 \ifx\bbl@languages\@undefined
979   \ifx\directlua\@undefined
980     \openin1 = language.def % TODO. Remove hardcoded number
981     \ifeof1
982       \closein1
983       \message{I couldn't find the file language.def}
984     \else
985       \closein1
986       \begingroup
987         \def\addlanguage#1#2#3#4#5{%
988           \expandafter\ifx\csname lang@#1\endcsname\relax\else
989             \global\expandafter\let\csname l@#1\endcsname
990               \csname lang@#1\endcsname
991           \fi}%
992         \def\uselanguage#1{%
993           \input language.def
994         \endgroup
995       \fi
996     \fi
997   \chardef\l@english\z@
998 \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`.

```
999 \def\addto#1#2{%
1000   \ifx#1\@undefined
1001     \def#1{#2}%
1002   \else
1003     \ifx#1\relax
```

```

1004     \def#1{#2}%
1005     \else
1006     {\toks@ \expandafter{#1#2}%
1007     \xdef#1{\the\toks@}}%
1008     \fi
1009 \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.

```

1010 \bbl@trace{Hooks}
1011 \newcommand\AddBabelHook[3][ ]{%
1012   \bbl@i funset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
1013   \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1014   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1015   \bbl@i funset{\bbl@ev@#2@#3@#1}%
1016   {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1017   {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1018   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1019 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1020 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1021 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1022 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1023   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1024   \def\bbl@elth##1{%
1025     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1026     \bbl@cs{ev@#2@#3}%
1027     \ifx\language\@undefined\else % Test required for Plain (?)
1028       \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1029       \def\bbl@elth##1{%
1030         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1031         \bbl@cs{ev@#2@#3}%
1032       \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).

```

1033 \def\bbl@evargs{,% <- don't delete this comma
1034   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1035   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1036   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1037   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1038   beforestart=0,language=2,beginndocument=1}
1039 \ifx\NewHook\@undefined\else % Test for Plain (?)
1040   \def\bbl@tempa#1=#2\@{ \NewHook{babel/#1}}
1041   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@{ }
1042 \fi

```

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

```

1043 \providecommand\PassOptionsToLocale[2]{%
1044   \bbl@csarg\bbl@add@list{passto@#2}{#1}}

```

4.7. Setting up language files

\LdfInit `\LdfInit` macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before.

At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a 'letter' during the processing of the file. We also save its name as the last called option, even if not loaded.

Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions with the `\let` primitive. Therefore we store its current catcode and restore it later on.

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing #2 through `string`. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\@undefined`.

If so, we call `\ldf@quit` to set the main language, restore the category code of the `@`-sign and call `\endinput`

When #2 was *not* a control sequence we construct one and compare it with `\relax`.

Finally we check `\originalTeX`.

```

1045 \bbl@trace{Macros for setting language files up}
1046 \def\bbl@ldfinit{%
1047   \let\bbl@screset\@empty
1048   \let\BabelStrings\bbl@opt@string
1049   \let\BabelOptions\@empty
1050   \let\BabelLanguages\relax
1051   \ifx\originalTeX\@undefined
1052     \let\originalTeX\@empty
1053   \else
1054     \originalTeX
1055   \fi}
1056 \def\LdfInit#1#2{%
1057   \chardef\atcatcode=\catcode`\@
1058   \catcode`\@=11\relax
1059   \chardef\eqcatcode=\catcode`\=
1060   \catcode`\==12\relax
1061   \expandafter\if\expandafter\@backslashchar
1062     \expandafter\@car\string#2\@nil
1063   \ifx#2\@undefined\else
1064     \ldf@quit{#1}%
1065   \fi
1066 \else
1067   \expandafter\ifx\csname#2\endcsname\relax\else
1068     \ldf@quit{#1}%
1069   \fi
1070 \fi
1071 \bbl@ldfinit}

```

\ldf@quit This macro interrupts the processing of a language definition file.

```

1072 \def\ldf@quit#1{%
1073   \expandafter\main@language\expandafter{#1}%
1074   \catcode`\@=\atcatcode \let\atcatcode\relax
1075   \catcode`\==\eqcatcode \let\eqcatcode\relax
1076   \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.

```

1077 \def\bbl@afterldf#1{%^A TODO. #1 is not used. Remove
1078   \bbl@afterlang
1079   \let\bbl@afterlang\relax
1080   \let\BabelModifiers\relax
1081   \let\bbl@screset\relax}%
1082 \def\ldf@finish#1{%
1083   \loadlocalcfg{#1}%
1084   \bbl@afterldf{#1}%
1085   \expandafter\main@language\expandafter{#1}%
1086   \catcode`\@=\atcatcode \let\atcatcode\relax
1087   \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 .

```
1088 \@onlypreamble\LdfInit
1089 \@onlypreamble\ldf@quit
1090 \@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.

```
1091 \def\main@language#1{%
1092   \def\bbl@main@language{#1}%
1093   \let\language\main@language
1094   \let\localename\bbl@main@language
1095   \let\mainlocalename\bbl@main@language
1096   \bbl@id@assign
1097   \bbl@patterns{\language}}
```

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

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

```
1098 \def\bbl@beforestart{%
1099   \def\@nolanerr##1{%
1100     \bbl@carg\chardef{l@##1}\z@
1101     \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1102   \bbl@usehooks{beforestart}{}%
1103   \global\let\bbl@beforestart\relax}
1104 \AtBeginDocument{%
1105   {\@nameuse{bbl@beforestart}}% Group!
1106   \if@filesw
1107     \providecommand\babel@aux[2]{}%
1108     \immediate\write\@mainaux{unexpanded{%
1109       \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1110     \immediate\write\@mainaux{string\@nameuse{bbl@beforestart}}%
1111   \fi
1112   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1113   \ifbbl@single % must go after the line above.
1114     \renewcommand\selectlanguage[1]{}%
1115     \renewcommand\foreignlanguage[2]{#2}%
1116     \global\let\babel@aux\@gobbletwo % Also as flag
1117   \fi}
1118 %
1119 \ifcase\bbl@engine\or
1120   \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1121 \fi
```

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

```
1122 \def\select@language@x#1{%
1123   \ifcase\bbl@select@type
1124     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1125   \else
1126     \select@language{#1}%
1127   \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.

```
1128 \bbl@trace{Shorthands}
1129 \def\bbl@withactive#1#2{%
```

```

1130 \begingroup
1131 \lccode`~=`#2\relax
1132 \lowercase{\endgroup#1~}}

```

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

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

```

1133 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1134 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1135 \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1136 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1137 \begingroup
1138 \catcode`#1\active
1139 \nfss@catcodes
1140 \ifnum\catcode`#1=\active
1141 \endgroup
1142 \bbl@add\nfss@catcodes{\@makeother#1}%
1143 \else
1144 \endgroup
1145 \fi
1146 \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).

```

1147 \def\bbl@active@def#1#2#3#4{%
1148 \namedef{#3#1}{%
1149 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1150 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1151 \else
1152 \bbl@afterfi\csname#2@sh@#1\endcsname
1153 \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.

```

1154 \long\namedef{#3@arg#1}##1{%
1155 \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1156 \bbl@afterelse\csname#4#1\endcsname##1%
1157 \else
1158 \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1159 \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.

```

1160 \def\initiate@active@char#1{%
1161   \bbl@ifunset{active@char\string#1}%
1162   {\bbl@withactive
1163     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1164   {}}

```

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).

```

1165 \def\@initiate@active@char#1#2#3{%
1166   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1167   \ifx#1\@undefined
1168     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1169   \else
1170     \bbl@csarg\let{oridef@#2}#1%
1171     \bbl@csarg\edef{oridef@#2}{%
1172       \let\noexpand#1%
1173       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1174   \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*).

```

1175   \ifx#1#3\relax
1176     \expandafter\let\csname normal@char#2\endcsname#3%
1177   \else
1178     \bbl@info{Making #2 an active character}%
1179     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1180     \@namedef{normal@char#2}{%
1181       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1182     \else
1183       \@namedef{normal@char#2}{#3}%
1184     \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).

```

1185   \bbl@restoreactive{#2}%
1186   \AtBeginDocument{%
1187     \catcode`#2\active
1188     \if@filesw
1189       \immediate\write\@mainaux{\catcode`\string#2\active}%
1190     \fi}%
1191   \expandafter\bbl@add@special\csname#2\endcsname
1192   \catcode`#2\active
1193 \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⟩).

```

1194 \let\bbl@tempa\@firstoftwo
1195 \if\string^#2%
1196   \def\bbl@tempa{\noexpand\textormath}%
1197 \else
1198   \ifx\bbl@mathnormal\@undefined\else
1199     \let\bbl@tempa\bbl@mathnormal
1200   \fi

```

```

1201 \fi
1202 \expandafter\edef\csname active@char#2\endcsname{%
1203   \bbl@tempa
1204   {\noexpand\if@safe@actives
1205     \noexpand\expandafter
1206     \expandafter\noexpand\csname normal@char#2\endcsname
1207     \noexpand\else
1208       \noexpand\expandafter
1209       \expandafter\noexpand\csname bbl@doactive#2\endcsname
1210     \noexpand\fi}%
1211   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1212 \bbl@csarg\edef{doactive#2}{%
1213   \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 $\backslash active@char \langle char \rangle$ is *one* control sequence!).

```

1214 \bbl@csarg\edef{active@#2}{%
1215   \noexpand\active@prefix\noexpand#1%
1216   \expandafter\noexpand\csname active@char#2\endcsname}%
1217 \bbl@csarg\edef{normal@#2}{%
1218   \noexpand\active@prefix\noexpand#1%
1219   \expandafter\noexpand\csname normal@char#2\endcsname}%
1220 \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.

```

1221 \bbl@active@def#2\user@group{user@active}{language@active}%
1222 \bbl@active@def#2\language@group{language@active}{system@active}%
1223 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1224 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1225   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1226 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1227   {\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 $\backslash prim@s$ as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```

1228 \if\string'#2%
1229   \let\prim@s\bbl@prim@s
1230   \let\active@math@prime#1%
1231 \fi
1232 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1233 << *More package options >> ≡
1234 \DeclareOption{math=active}{}
1235 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1236 << /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*.


```

1237 \@ifpackagewith{babel}{KeepShorthandsActive}%
1238 {\let\bbl@restoreactive\@gobble}%
1239 {\def\bbl@restoreactive#1{%
1240   \bbl@exp{%
1241     \\AfterBabelLanguage\\CurrentOption
1242     {\catcode`#1=\the\catcode`#1\relax}%
1243     \\AtEndOfPackage
1244     {\catcode`#1=\the\catcode`#1\relax}}}%
1245   \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.

```

1246 \def\bbl@sh@select#1#2{%
1247   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1248     \bbl@afterelse\bbl@scndcs
1249   \else
1250     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1251   \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.

```

1252 \begingroup
1253 \bbl@ifunset{ifincsname}%^^A Ugly. Correct? Only Plain?
1254 {\gdef\active@prefix#1{%
1255   \ifx\protect\@typeset@protect
1256   \else
1257     \ifx\protect\@unexpandable@protect
1258       \noexpand#1%
1259     \else
1260       \protect#1%
1261     \fi
1262   \expandafter\@gobble
1263   \fi}}
1264 {\gdef\active@prefix#1{%
1265   \ifincsname
1266     \string#1%
1267     \expandafter\@gobble
1268   \else
1269     \ifx\protect\@typeset@protect
1270     \else
1271       \ifx\protect\@unexpandable@protect
1272         \noexpand#1%
1273       \else
1274         \protect#1%
1275       \fi
1276     \expandafter\expandafter\expandafter\@gobble
1277     \fi
1278   \fi}}
1279 \endgroup

```

if@safe@actives In some circumstances it is necessary to be able to reset the shorthand to its ‘normal’ value (usually the character with catcode ‘other’) on the fly. For this purpose the switch @safe@actives is available. The setting of this switch should be checked in the first level expansion of \active@char<char>. When this expansion mode is active (with \@safe@activetrue), something like "13"13 becomes "12"12 in an \edef (in other words, shorthands are \string'ed). This contrasts

with `\protected@edef`, where catcodes are always left unchanged. Once converted, they can be used safely even after this expansion mode is deactivated (with `\@safe@activefalse`).

```
1280 \newif\if@safe@actives
1281 \@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.

```
1282 \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`.

```
1283 \chardef\bbl@activated\z@
1284 \def\bbl@activate#1{%
1285   \chardef\bbl@activated\@ne
1286   \bbl@withactive{\expandafter\let\expandafter}#1%
1287   \csname bbl@active@\string#1\endcsname}
1288 \def\bbl@deactivate#1{%
1289   \chardef\bbl@activated\tw@
1290   \bbl@withactive{\expandafter\let\expandafter}#1%
1291   \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs

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

```
1292 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1293 \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.

```
1294 \def\babel@texpdf#1#2#3#4{%
1295   \ifx\texorpdfstring\undefined
1296     \textormath{#1}{#3}%
1297   \else
1298     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1299     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1300   \fi}
1301 %
1302 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1303 \def\@decl@short#1#2#3\@nil#4{%
1304   \def\bbl@tempa{#3}%
1305   \ifx\bbl@tempa\@empty
1306     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1307     \bbl@ifunset{#1@sh@\string#2@}{}%
1308     {\def\bbl@tempa{#4}%
1309      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1310       \else
1311         \bbl@info
1312           {Redefining #1 shorthand \string#2\}%
1313           in language \CurrentOption}%
1314     \fi}%
1315   \@namedef{#1@sh@\string#2@}{#4}%
```

```

1316 \else
1317   \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1318   \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1319   {\def\bbl@tempa{#4}%
1320    \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1321    \else
1322      \bbl@info
1323      {Redefining #1 shorthand \string#2\string#3\\%
1324       in language \CurrentOption}%
1325      \fi}%
1326   \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1327 \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.

```

1328 \def\textormath{%
1329   \ifmmode
1330     \expandafter\@secondoftwo
1331   \else
1332     \expandafter\@firstoftwo
1333   \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’.

```

1334 \def\user@group{user}
1335 \def\language@group{english} %^^A I don't like defaults
1336 \def\system@group{system}

```

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

```

1337 \def\usesshorthands{%
1338   \@ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1339 \def\bbl@usesesh@s#1{%
1340   \bbl@usesesh@x
1341   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1342   {#1}}
1343 \def\bbl@usesesh@x#1#2{%
1344   \bbl@ifshorthand{#2}%
1345   {\def\user@group{user}%
1346    \initiate@active@char{#2}%
1347    #1%
1348    \bbl@activate{#2}}%
1349   {\bbl@error{shorthand-is-off}{#2}{}}}

```

\defineshorthand Currently we only support two groups of user level shorthands, named internally `user` and `user@(\language)` (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of `\defineshorthand`) a new level is inserted for it (`user@generic`, done by `\bbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1350 \def\user@language@group{user@\language@group}
1351 \def\bbl@set@user@generic#1#2{%
1352   \bbl@ifunset{user@generic@active#1}%
1353   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1354    \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1355    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1356      \expandafter\noexpand\csname normal@char#1\endcsname}%

```

```

1357 \expandafter\edef\csname#2@sh@#1\string\protect@endcsname{%
1358 \expandafter\noexpand\csname user@active#1@endcsname}}%
1359 \@empty}
1360 \newcommand\defineshorthand[3][user]{%
1361 \edef\bbl@tempa{\zap@space#1 \@empty}%
1362 \bbl@for\bbl@tempb\bbl@tempa{%
1363 \if*\expandafter\@car\bbl@tempb\@nil
1364 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1365 \@expandtwoargs
1366 \bbl@setuser@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1367 \fi
1368 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

\languageshorthands A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed.

```

1369 \def\languageshorthands#1{\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".

```

1370 \def\aliasshorthand#1#2{%
1371 \bbl@ifshorthand{#2}%
1372 {\expandafter\ifx\csname active@char\string#2@endcsname\relax
1373 \ifx\document\@notprerr
1374 \@notshorthand{#2}%
1375 \else
1376 \initiate@active@char{#2}%
1377 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1378 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1379 \bbl@activate{#2}%
1380 \fi
1381 \fi}%
1382 {\bbl@error{shorthand-is-off}{#2}{}}}

```

\@notshorthand

```

1383 \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.

```

1384 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1385 \DeclareRobustCommand*\shorthandoff{%
1386 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1387 \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.

```

1388 \def\bbl@switch@sh#1#2{%
1389 \ifx#2\@nnil\else
1390 \bbl@ifunset{\bbl@active@\string#2}%
1391 {\bbl@error{not-a-shorthand-b}{#2}{}}%
1392 {\ifcase#1% off, on, off*
1393 \catcode`#212\relax

```

```

1394 \or
1395 \catcode`#2\active
1396 \bbl@ifunset{bbl@shdef@\string#2}%
1397 {}%
1398 {\bbl@withactive{\expandafter\let\expandafter}#2%
1399 \csname bbl@shdef@\string#2\endcsname
1400 \bbl@csarg\let{shdef@\string#2}\relax}%
1401 \ifcase\bbl@activated\or
1402 \bbl@activate{#2}%
1403 \else
1404 \bbl@deactivate{#2}%
1405 \fi
1406 \or
1407 \bbl@ifunset{bbl@shdef@\string#2}%
1408 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1409 {}%
1410 \csname bbl@oricat@\string#2\endcsname
1411 \csname bbl@oridef@\string#2\endcsname
1412 \fi}%
1413 \bbl@afterfi\bbl@switch@sh#1%
1414 \fi}

```

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

```

1415 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1416 \def\bbl@putsh#1{%
1417 \bbl@ifunset{bbl@active@\string#1}%
1418 {\bbl@putsh@i#1\@empty\@nnil}%
1419 {\csname bbl@active@\string#1\endcsname}}
1420 \def\bbl@putsh@i#1#2\@nnil{%
1421 \csname\language@group @sh@\string#1@%
1422 \ifx\@empty#2\else\string#2@\fi\endcsname}
1423 %
1424 \ifx\bbl@opt@shorthands\@nnil\else
1425 \let\bbl@s@initiate@active@char\initiate@active@char
1426 \def\initiate@active@char#1{%
1427 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1428 \let\bbl@s@switch@sh\bbl@switch@sh
1429 \def\bbl@switch@sh#1#2{%
1430 \ifx#2\@nnil\else
1431 \bbl@afterfi
1432 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1433 \fi}
1434 \let\bbl@s@activate\bbl@activate
1435 \def\bbl@activate#1{%
1436 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1437 \let\bbl@s@deactivate\bbl@deactivate
1438 \def\bbl@deactivate#1{%
1439 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1440 \fi

```

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

```

1441 \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.

```

1442 \def\bbl@prim@s{%
1443 \prime\futurelet\@let@token\bbl@pr@m@s}
1444 \def\bbl@if@primes#1#2{%

```

```

1445 \ifx#1\@let@token
1446 \expandafter\@firstoftwo
1447 \else\ifx#2\@let@token
1448 \bbl@afterelse\expandafter\@firstoftwo
1449 \else
1450 \bbl@afterfi\expandafter\@secondoftwo
1451 \fi\fi}
1452 \begingroup
1453 \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
1454 \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
1455 \lowercase{%
1456 \gdef\bbl@pr@ms{%
1457 \bbl@if@primes''%
1458 \pr@@s
1459 {\bbl@if@primes*\pr@@t\egroup}}
1460 \endgroup

```

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

```

1461 \initiate@active@char{~}
1462 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1463 \bbl@activate{~}

```

\OT1dqpos

\T1dqpos The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the `\f@encoding` macro. Therefore we define two macros here to store the position of the character in these encodings.

```

1464 \expandafter\def\csname OT1dqpos\endcsname{127}
1465 \expandafter\def\csname T1dqpos\endcsname{4}

```

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

```

1466 \ifx\f@encoding\undefined
1467 \def\f@encoding{OT1}
1468 \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.

```

1469 \bbl@trace{Language attributes}
1470 \newcommand\languageattribute[2]{%
1471 \def\bbl@tempc{#1}%
1472 \bbl@fixname\bbl@tempc
1473 \bbl@iflanguage\bbl@tempc{%
1474 \bbl@vforeach{#2}{%

```

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

```

1475 \ifx\bbl@known@attribs\undefined
1476 \in@false
1477 \else
1478 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1479 \fi

```

```

1480 \ifin@
1481 \bbl@warning{%
1482     You have more than once selected the attribute '##1'\%
1483     for language #1. Reported}%
1484 \else

```

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

```

1485 \bbl@exp{%
1486     \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1487 \edef\bbl@tempa{\bbl@tempc-##1}%
1488 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1489 {\csname\bbl@tempc @attr##1\endcsname}%
1490 {\@attrerr{\bbl@tempc}{##1}}%
1491 \fi}}}
1492 \@onlypreamble\languageattribute

```

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

```

1493 \newcommand*{\@attrerr}[2]{%
1494     \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}`.

```

1495 \def\bbl@declare@ttribute#1#2#3{%
1496     \bbl@xin@{#2,}{\BabelModifiers,}%
1497     \ifin@
1498         \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1499     \fi
1500     \bbl@add@list\bbl@attributes{#1-#2}%
1501     \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.

```

1502 \def\bbl@ifattributeset#1#2#3#4{%
1503     \ifx\bbl@known@attribs\@undefined
1504         \in@false
1505     \else
1506         \bbl@xin@{#1-#2,}{\bbl@known@attribs,}%
1507     \fi
1508     \ifin@
1509         \bbl@afterelse#3%
1510     \else
1511         \bbl@afterfi#4%
1512     \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.

```

1513 \def\bbl@ifknown@ttrib#1#2{%
1514     \let\bbl@tempa\@secondoftwo
1515     \bbl@loopx\bbl@tempb{#2}{%
1516         \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{#1,}%
1517     \ifin@

```

```

1518     \let\bbl@tempa\@firstoftwo
1519     \else
1520     \fi}%
1521     \bbl@tempa}

```

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

```

1522 \def\bbl@clear@ttribs{%
1523   \ifx\bbl@attributes\undefined\else
1524     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1525       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1526     \let\bbl@attributes\undefined
1527   \fi}
1528 \def\bbl@clear@ttrib#1-#2.{%
1529   \expandafter\let\csname#1@attr#2\endcsname\@undefined}
1530 \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.

```

1531 \bbl@trace{Macros for saving definitions}
1532 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1533 \newcount\babel@savecnt
1534 \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.

```

1535 \def\babel@save#1{%
1536   \def\bbl@tempa{{, #1,}}% Clumsy, for Plain
1537   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1538     \expandafter{\expandafter,\bbl@savextras,}}%
1539   \expandafter\in@\bbl@tempa
1540   \ifin@ \else
1541     \bbl@add\bbl@savextras{, #1,}%
1542     \bbl@carg\let{\babel@number\babel@savecnt}#1\relax
1543     \toks@\expandafter{\originalTeX\let#1=}%
1544     \bbl@exp{%
1545       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1546     \advance\babel@savecnt@ne
1547   \fi}
1548 \def\babel@savevariable#1{%
1549   \toks@\expandafter{\originalTeX #1=}%
1550   \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`.

```
1551 \def\bbl@redefine#1{%
1552   \edef\bbl@tempa{\bbl@stripslash#1}%
1553   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1554   \expandafter\def\csname\bbl@tempa\endcsname}
1555 \@onlypreamble\bbl@redefine
```

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

```
1556 \def\bbl@redefine@long#1{%
1557   \edef\bbl@tempa{\bbl@stripslash#1}%
1558   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1559   \long\expandafter\def\csname\bbl@tempa\endcsname}
1560 \@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`.

```
1561 \def\bbl@redefineroobust#1{%
1562   \edef\bbl@tempa{\bbl@stripslash#1}%
1563   \bbl@ifunset{\bbl@tempa\space}%
1564     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1565      \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1566     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1567     \@namedef{\bbl@tempa\space}}
1568 \@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.

```
1569 \def\bbl@frenchspacing{%
1570   \ifnum\the\sfcode\.\=\@m
1571     \let\bbl@nonfrenchspacing\relax
1572   \else
1573     \frenchspacing
1574     \let\bbl@nonfrenchspacing\nonfrenchspacing
1575   \fi}
1576 \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.

```
1577 \let\bbl@elt\relax
1578 \edef\bbl@fs@chars{%
1579   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1580   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1581   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1582 \def\bbl@pre@fs{%
1583   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1584   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1585   \def\bbl@post@fs{%
1586     \bbl@save@sfcodes
1587     \edef\bbl@tempa{\bbl@cl{frspc}}%
1588     \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1589   }
```

```

1589 \if u\bbl@tempa      % do nothing
1590 \else\if n\bbl@tempa  % non french
1591   \def\bbl@elt##1##2##3{%
1592     \ifnum\sfcode`##1=##2\relax
1593       \babel@savevariable{\sfcode`##1}%
1594       \sfcode`##1=##3\relax
1595     \fi}%
1596   \bbl@fs@chars
1597 \else\if y\bbl@tempa   % french
1598   \def\bbl@elt##1##2##3{%
1599     \ifnum\sfcode`##1=##3\relax
1600       \babel@savevariable{\sfcode`##1}%
1601       \sfcode`##1=##2\relax
1602     \fi}%
1603   \bbl@fs@chars
1604 \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.

```

1605 \bbl@trace{Hyphens}
1606 \@onlypreamble\babelhyphenation
1607 \AtEndOfPackage{%
1608   \newcommand\babelhyphenation[2][\@empty]{%
1609     \ifx\bbl@hyphenation@\relax
1610       \let\bbl@hyphenation@\@empty
1611     \fi
1612     \ifx\bbl@hyphlist\@empty\else
1613       \bbl@warning{%
1614         You must not intermingle \string\selectlanguage\space and\\%
1615         \string\babelhyphenation\space or some exceptions will not\\%
1616         be taken into account. Reported}%
1617       \fi
1618       \ifx\@empty#1%
1619         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1620       \else
1621         \bbl@vforeach{#1}{%
1622           \def\bbl@tempa{##1}%
1623           \bbl@fixname\bbl@tempa
1624           \bbl@iflanguage\bbl@tempa{%
1625             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1626               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1627               }{%
1628                 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1629                 #2}}}%
1630         \fi}}

```

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

```

1631 \ifx\NewDocumentCommand\@undefined\else
1632   \NewDocumentCommand\babelhyphenmins{sommo}{%
1633     \IfNoValueTF{#2}%
1634       {\protected@edef\bbl@hyphenmins@{\set@hyphenmins{#3}{#4}}}%
1635       \IfValueT{#5}{%
1636         \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1637       \IfBooleanT{#1}{%
1638         \lefthyphenmin=#3\relax
1639         \righthyphenmin=#4\relax
1640         \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1641     {\edef\bbl@tempb{\zap@space#2 \@empty}%

```

```

1642 \bbl@for\bbl@tempa\bbl@tempb{%
1643 \namedef\bbl@hyphenmins@bbl@tempa{\set@hyphenmins{#3}{#4}}%
1644 \IfValueT{#5}{%
1645 \namedef\bbl@hyphenatmin@bbl@tempa{\hyphenationmin=#5\relax}}%
1646 \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{}}{}}
1647 \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.

```

1648 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1649 \def\bbl@t@one{T1}
1650 \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`.

```

1651 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1652 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1653 \def\bbl@hyphen{%
1654 \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1655 \def\bbl@hyphen@i#1#2{%
1656 \bbl@iifunset\bbl@hy@#1#2\@empty}%
1657 {\csname bbl@#1usehyphen\endcsname\discretionary{#2}{#2}}%
1658 {\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.

```

1659 \def\bbl@usehyphen#1{%
1660 \leavevmode
1661 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1662 \nobreak\hskip\z@skip}
1663 \def\bbl@@usehyphen#1{%
1664 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1665 \def\bbl@hyphenchar{%
1666 \ifnum\hyphenchar\font=\m@ne
1667 \babelnullhyphen
1668 \else
1669 \char\hyphenchar\font
1670 \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.

```

1671 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}
1672 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}
1673 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1674 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1675 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1676 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1677 \def\bbl@hy@repeat{%
1678 \bbl@usehyphen%
1679 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1680 \def\bbl@hy@@repeat{%
1681 \bbl@usehyphen%
1682 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}

```

```

1683 \def\bbl@hy@empty{\hskip\z@skip}
1684 \def\bbl@hy@@empty{\discretionary{}{}{}}

```

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

```

1685 \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.

```

1686 \bbl@trace{Multiencoding strings}
1687 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1688 << *More package options >> ≡
1689 \DeclareOption{nocase}{}
1690 << /More package options >>

```

The following package options control the behavior of \SetString.

```

1691 << *More package options >> ≡
1692 \let\bbl@opt@strings\@nnil % accept strings=value
1693 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1694 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1695 \def\BabelStringsDefault{generic}
1696 << /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.

```

1697 \@onlypreamble\StartBabelCommands
1698 \def\StartBabelCommands{%
1699   \begingroup
1700   \@tempcnta="7F
1701   \def\bbl@tempa{%
1702     \ifnum\@tempcnta>"FF\else
1703       \catcode\@tempcnta=11
1704       \advance\@tempcnta\@ne
1705       \expandafter\bbl@tempa
1706     \fi}%
1707   \bbl@tempa
1708   <@Macros local to BabelCommands@>
1709   \def\bbl@provstring##1##2{%
1710     \providecommand##1{##2}%
1711     \bbl@tglobal##1}%
1712   \global\let\bbl@scafter\@empty
1713   \let\StartBabelCommands\bbl@startcmds
1714   \ifx\BabelLanguages\relax
1715     \let\BabelLanguages\CurrentOption
1716   \fi
1717   \begingroup
1718   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1719   \StartBabelCommands}
1720 \def\bbl@startcmds{%
1721   \ifx\bbl@screset\@nnil\else
1722     \bbl@usehooks{stopcommands}{}%
1723   \fi
1724   \endgroup

```

```

1725 \begingroup
1726 \@ifstar
1727   {\ifx\bbbl@opt@strings\@nnil
1728     \let\bbbl@opt@strings\BabelStringsDefault
1729     \fi
1730     \bbbl@startcmds@i}%
1731   \bbbl@startcmds@i}
1732 \def\bbbl@startcmds@i#1#2{%
1733   \edef\bbbl@L{\zap@space#1 \@empty}%
1734   \edef\bbbl@G{\zap@space#2 \@empty}%
1735   \bbbl@startcmds@ii}
1736 \let\bbbl@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.

```

1737 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1738   \let\SetString\@gobbletwo
1739   \let\bbbl@stringdef\@gobbletwo
1740   \let\AfterBabelCommands\@gobble
1741   \ifx\@empty#1%
1742     \def\bbbl@sc@label{generic}%
1743     \def\bbbl@encstring##1##2{%
1744       \ProvideTextCommandDefault##1{##2}%
1745       \bbbl@tglobal##1%
1746       \expandafter\bbbl@tglobal\csname\string?\string##1\endcsname}%
1747     \let\bbbl@sctest\in@true
1748   \else
1749     \let\bbbl@sc@charset\space % <- zapped below
1750     \let\bbbl@sc@fontenc\space % <- " "
1751     \def\bbbl@tempa##1=##2\@nil{%
1752       \bbbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1753     \bbbl@vforeach{label=#1}{\bbbl@tempa##1\@nil}%
1754     \def\bbbl@tempa##1 ##2{% space -> comma
1755       ##1%
1756       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
1757     \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
1758     \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
1759     \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
1760     \def\bbbl@encstring##1##2{%
1761       \bbbl@foreach\bbbl@sc@fontenc{%
1762         \bbbl@ifunset{T@####1}%
1763         }%
1764         {\ProvideTextCommand##1{####1}{##2}%
1765         \bbbl@tglobal##1%
1766         \expandafter
1767         \bbbl@tglobal\csname####1\string##1\endcsname}}}%
1768     \def\bbbl@sctest{%
1769       \bbbl@xin{\bbbl@opt@strings,}{,\bbbl@sc@label,\bbbl@sc@fontenc,}}%
1770   \fi
1771   \ifx\bbbl@opt@strings\@nnil % i.e., no strings key -> defaults
1772   \else\ifx\bbbl@opt@strings\relax % i.e., strings=encoded
1773     \let\AfterBabelCommands\bbbl@aftercmds
1774     \let\SetString\bbbl@setstring
1775     \let\bbbl@stringdef\bbbl@encstring
1776   \else % i.e., strings=value
1777     \bbbl@sctest

```

```

1778 \ifin@
1779 \let\AfterBabelCommands\bbbl@aftercmds
1780 \let\SetString\bbbl@setstring
1781 \let\bbbl@stringdef\bbbl@provstring
1782 \fi\fi\fi
1783 \bbbl@scswitch
1784 \ifx\bbbl@G\@empty
1785 \def\SetString##1##2{%
1786 \bbbl@error{missing-group}{##1}{}}}%
1787 \fi
1788 \ifx\@empty#1%
1789 \bbbl@usehooks{defaultcommands}{}%
1790 \else
1791 \@expandtwoargs
1792 \bbbl@usehooks{encodedcommands}{\bbbl@sc@charset}\bbbl@sc@fontenc}}%
1793 \fi}

```

There are two versions of `\bbbl@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 (`\bbbl@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 `\bbbl@forlang` loops `\bbbl@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 `\bbbl@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).

```

1794 \def\bbbl@forlang#1#2{%
1795 \bbbl@for#1\bbbl@L{%
1796 \bbbl@xin@{, #1, }{, \BabelLanguages,}%
1797 \ifin@#2\relax\fi}}
1798 \def\bbbl@scswitch{%
1799 \bbbl@forlang\bbbl@tempa{%
1800 \ifx\bbbl@G\@empty\else
1801 \ifx\SetString\@gobbletwo\else
1802 \edef\bbbl@GL{\bbbl@G\bbbl@tempa}%
1803 \bbbl@xin@{, \bbbl@GL, }{, \bbbl@screset,}%
1804 \ifin@else
1805 \global\expandafter\let\csname\bbbl@GL\endcsname\@undefined
1806 \xdef\bbbl@screset{\bbbl@screset, \bbbl@GL}%
1807 \fi
1808 \fi
1809 \fi}}
1810 \AtEndOfPackage{%
1811 \def\bbbl@forlang#1#2{\bbbl@for#1\bbbl@L{\bbbl@ifunset{date#1}{}}{#2}}}%
1812 \let\bbbl@scswitch\relax}
1813 \@onlypreamble\EndBabelCommands
1814 \def\EndBabelCommands{%
1815 \bbbl@usehooks{stopcommands}{}%
1816 \endgroup
1817 \endgroup
1818 \bbbl@scafter}
1819 \let\bbbl@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.

```

1820 \def\bbbl@setstring#1#2{% e.g., \prefacename{<string>}
1821 \bbbl@forlang\bbbl@tempa{%
1822 \edef\bbbl@LC{\bbbl@tempa\bbbl@stripslash#1}%
1823 \bbbl@ifunset{\bbbl@LC}% e.g., \germanchaptername

```

```

1824      {\bbl@exp{%
1825        \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
1826      }%
1827      \def\BabelString{#2}%
1828      \bbl@usehooks{stringprocess}{}%
1829      \expandafter\bbl@stringdef
1830      \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`.

```
1831 \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.

```

1832 <<*Macros local to BabelCommands>> ≡
1833 \def\SetStringLoop##1##2{%
1834   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1835   \count@\z@
1836   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1837     \advance\count@\@ne
1838     \toks@\expandafter{\bbl@tempa}%
1839     \bbl@exp{%
1840       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1841       \count@=\the\count@\relax}}}%
1842 <</Macros local to BabelCommands>>

```

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

```

1843 \def\bbl@aftercmds#1{%
1844   \toks@\expandafter{\bbl@scafter#1}%
1845   \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.

```

1846 <<*Macros local to BabelCommands>> ≡
1847 \newcommand\SetCase[3][{}]{%
1848   \def\bbl@tempa####1####2{%
1849     \ifx####1\@empty\else
1850       \bbl@carg\bbl@add{extras\CurrentOption}{%
1851         \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1852         \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1853         \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1854         \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1855       \expandafter\bbl@tempa
1856     \fi}%
1857   \bbl@tempa##1\@empty\@empty
1858   \bbl@carg\bbl@toglobal{extras\CurrentOption}}%
1859 <</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.

```

1860 <<*Macros local to BabelCommands>> ≡
1861 \newcommand\SetHyphenMap[1]{%
1862   \bbl@forlang\bbl@tempa{%
1863     \expandafter\bbl@stringdef
1864     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1865 <</Macros local to BabelCommands>>

```

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

```

1866 \newcommand\BabelLower[2]{% one to one.
1867   \ifnum\lccode#1=#2\else

```

```

1868 \babel@savevariable{\lccode#1}%
1869 \lccode#1=#2\relax
1870 \fi}
1871 \newcommand\BabelLowerMM[4]{% many-to-many
1872 \@tempcnta=#1\relax
1873 \@tempcntb=#4\relax
1874 \def\bbl@tempa{%
1875 \ifnum\@tempcnta>#2\else
1876 \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1877 \advance\@tempcnta#3\relax
1878 \advance\@tempcntb#3\relax
1879 \expandafter\bbl@tempa
1880 \fi}%
1881 \bbl@tempa}
1882 \newcommand\BabelLowerM0[4]{% many-to-one
1883 \@tempcnta=#1\relax
1884 \def\bbl@tempa{%
1885 \ifnum\@tempcnta>#2\else
1886 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1887 \advance\@tempcnta#3
1888 \expandafter\bbl@tempa
1889 \fi}%
1890 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1891 <<{*More package options}>> ≡
1892 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1893 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1894 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1895 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1896 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1897 <</More package options>>

```

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

```

1898 \AtEndOfPackage{%
1899 \ifx\bbl@opt@hyphenmap\@undefined
1900 \bbl@xin@{,}{\bbl@language@opts}%
1901 \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1902 \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.

```

1903 \newcommand\setlocalecaption{%^^A Catch typos.
1904 \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1905 \def\bbl@setcaption@x#1#2#3{% language caption-name string
1906 \bbl@trim@def\bbl@tempa{#2}%
1907 \bbl@xin@{.template}{\bbl@tempa}%
1908 \ifin@
1909 \bbl@ini@captions@template{#3}{#1}%
1910 \else
1911 \edef\bbl@tempd{%
1912 \expandafter\expandafter\expandafter
1913 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1914 \bbl@xin@
1915 {\expandafter\string\csname #2name\endcsname}%
1916 {\bbl@tempd}%
1917 \ifin@ % Renew caption
1918 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
1919 \ifin@
1920 \bbl@exp{%
1921 \\bbl@ifsamestring{\bbl@tempa}{\language name}%

```



```

1922         {\bbl@scset\<#2name>\<#1#2name>}%
1923     }%
1924     \else % Old way converts to new way
1925         \bbl@ifunset{#1#2name}%
1926         {\bbl@exp{%
1927             \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1928             \\bbl@ifsamestring{\bbl@tempa}{\language}%
1929             {\def\<#2name>{\<#1#2name>}}%
1930             }%
1931         }%
1932     \fi
1933 \else
1934     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1935     \ifin@ % New way
1936         \bbl@exp{%
1937             \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}}%
1938             \\bbl@ifsamestring{\bbl@tempa}{\language}%
1939             {\bbl@scset\<#2name>\<#1#2name>}}%
1940             }%
1941         \else % Old way, but defined in the new way
1942             \bbl@exp{%
1943                 \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1944                 \\bbl@ifsamestring{\bbl@tempa}{\language}%
1945                 {\def\<#2name>{\<#1#2name>}}%
1946                 }%
1947             \fi%
1948         \fi
1949         \@namedef{#1#2name}{#3}%
1950         \toks@{\expandafter{\bbl@captionslist}}%
1951         \bbl@exp{\in{\<#2name>}{\the\toks@}}%
1952         \ifin@ \else
1953             \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
1954             \bbl@tglobal\bbl@captionslist
1955         \fi
1956     \fi}
1957 %^^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.

```

1958 \bbl@trace{Macros related to glyphs}
1959 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
1960     \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@
1961     \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.

```

1962 \def\save@sf@q#1{\leavevmode
1963     \begingroup
1964     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1965     \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.

```

1966 \ProvideTextCommand{\quotedblbase}{OT1}{%

```

```

1967 \save@sf@q{\set@low@box{\textquotedblright\}}%
1968 \box\z@\kern-.04em\bbbl@allowhyphens}}

```

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

```

1969 \ProvideTextCommandDefault{\quotedblbase}{%
1970 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1971 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1972 \save@sf@q{\set@low@box{\textquoteright\}}%
1973 \box\z@\kern-.04em\bbbl@allowhyphens}}

```

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

```

1974 \ProvideTextCommandDefault{\quotesinglbase}{%
1975 \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

1976 \ProvideTextCommand{\guillemetleft}{OT1}{%
1977 \ifmmode
1978 \ll
1979 \else
1980 \save@sf@q{\nobreak
1981 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbbl@allowhyphens}%
1982 \fi}
1983 \ProvideTextCommand{\guillemetright}{OT1}{%
1984 \ifmmode
1985 \gg
1986 \else
1987 \save@sf@q{\nobreak
1988 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbbl@allowhyphens}%
1989 \fi}
1990 \ProvideTextCommand{\guillemotleft}{OT1}{%
1991 \ifmmode
1992 \ll
1993 \else
1994 \save@sf@q{\nobreak
1995 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbbl@allowhyphens}%
1996 \fi}
1997 \ProvideTextCommand{\guillemotright}{OT1}{%
1998 \ifmmode
1999 \gg
2000 \else
2001 \save@sf@q{\nobreak
2002 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbbl@allowhyphens}%
2003 \fi}

```

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

```

2004 \ProvideTextCommandDefault{\guillemetleft}{%
2005 \UseTextSymbol{OT1}{\guillemetleft}}
2006 \ProvideTextCommandDefault{\guillemetright}{%
2007 \UseTextSymbol{OT1}{\guillemetright}}
2008 \ProvideTextCommandDefault{\guillemotleft}{%
2009 \UseTextSymbol{OT1}{\guillemotleft}}
2010 \ProvideTextCommandDefault{\guillemotright}{%
2011 \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2012 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2013   \ifmmode
2014     <%
2015   \else
2016     \save@sf@q{\nobreak
2017       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2018   \fi}
2019 \ProvideTextCommand{\guilsinglright}{OT1}{%
2020   \ifmmode
2021     >%
2022   \else
2023     \save@sf@q{\nobreak
2024       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2025   \fi}

```

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

```

2026 \ProvideTextCommandDefault{\guilsinglleft}{%
2027   \UseTextSymbol{OT1}{\guilsinglleft}}
2028 \ProvideTextCommandDefault{\guilsinglright}{%
2029   \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.

```

2030 \DeclareTextCommand{\ij}{OT1}{%
2031   i\kern-0.02em\bbl@allowhyphens j}
2032 \DeclareTextCommand{\IJ}{OT1}{%
2033   I\kern-0.02em\bbl@allowhyphens J}
2034 \DeclareTextCommand{\ij}{T1}{\char188}
2035 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2036 \ProvideTextCommandDefault{\ij}{%
2037   \UseTextSymbol{OT1}{\ij}}
2038 \ProvideTextCommandDefault{\IJ}{%
2039   \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).

```

2040 \def\crrtic@{\hrule height0.1ex width0.3em}
2041 \def\crrtic@{\hrule height0.1ex width0.33em}
2042 \def\ddj@{%
2043   \setbox0\hbox{d}\dimen@=\ht0
2044   \advance\dimen@lex
2045   \dimen@.45\dimen@
2046   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2047   \advance\dimen@ii.5ex
2048   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2049 \def\DDJ@{%
2050   \setbox0\hbox{D}\dimen@=.55\ht0
2051   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2052   \advance\dimen@ii.15ex % correction for the dash position
2053   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2054   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2055   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2056 %

```

```

2057 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2058 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2059 \ProvideTextCommandDefault{\dj}{%
2060   \UseTextSymbol{OT1}{\dj}}
2061 \ProvideTextCommandDefault{\DJ}{%
2062   \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.

```

2063 \DeclareTextCommand{\SS}{OT1}{SS}
2064 \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.

```

2065 \ProvideTextCommandDefault{\glq}{%
2066   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

```

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

```

2067 \ProvideTextCommand{\grq}{T1}{%
2068   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2069 \ProvideTextCommand{\grq}{TU}{%
2070   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2071 \ProvideTextCommand{\grq}{OT1}{%
2072   \save@sf@q{\kern-.0125em
2073     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2074     \kern.07em\relax}}
2075 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2076 \ProvideTextCommandDefault{\glqq}{%
2077   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2078 \ProvideTextCommand{\grqq}{T1}{%
2079   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2080 \ProvideTextCommand{\grqq}{TU}{%
2081   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2082 \ProvideTextCommand{\grqq}{OT1}{%
2083   \save@sf@q{\kern-.07em
2084     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2085     \kern.07em\relax}}
2086 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2087 \ProvideTextCommandDefault{\flq}{%
2088   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2089 \ProvideTextCommandDefault{\frq}{%
2090   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```
2091 \ProvideTextCommandDefault{\flqq}{%
2092   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2093 \ProvideTextCommandDefault{\frqq}{%
2094   \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).

```
2095 \def\umlauthigh{%
2096   \def\bbl@umlauta##1{\leavevmode\bgroup%
2097     \accent\csname\fontencoding dqpos\endcsname
2098     ##1\bbl@allowhyphens\egroup}%
2099   \let\bbl@umlaute\bbl@umlauta}
2100 \def\umlautlow{%
2101   \def\bbl@umlauta{\protect\lower@umlaut}}
2102 \def\umlautelow{%
2103   \def\bbl@umlaute{\protect\lower@umlaut}}
2104 \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.

```
2105 \expandafter\ifx\csname U@D\endcsname\relax
2106   \csname newdimen\endcsname U@D
2107 \fi
```

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

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

```
2108 \def\lower@umlaut#1{%
2109   \leavevmode\bgroup
2110   \U@D lex%
2111   {\setbox\z@\hbox{%
2112     \char\csname\fontencoding dqpos\endcsname}%
2113     \dimen@ -.45ex\advance\dimen@\ht\z@
2114     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2115   \accent\csname\fontencoding dqpos\endcsname
2116   \fontdimen5\font\U@D #1%
2117   \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).

```
2118 \AtBeginDocument{%
2119   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2120   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2121   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
```

```

2122 \DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaut{i}}%
2123 \DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlaut{o}}%
2124 \DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlaut{u}}%
2125 \DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlaut{A}}%
2126 \DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaut{E}}%
2127 \DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaut{I}}%
2128 \DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlaut{O}}%
2129 \DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlaut{U}}%

```

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

```

2130 \ifx\l@english\@undefined
2131 \chardef\l@english\z@
2132 \fi
2133 % The following is used to cancel rules in ini files (see Amharic).
2134 \ifx\l@unhyphenated\@undefined
2135 \newlanguage\l@unhyphenated
2136 \fi

```

4.16. Layout

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

```

2137 \bbl@trace{Bidi layout}
2138 \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.

```

2139 \bbl@trace{Input engine specific macros}
2140 \ifcase\bbl@engine
2141 \input txtbabel.def
2142 \or
2143 \input luababel.def
2144 \or
2145 \input xebabel.def
2146 \fi
2147 \providecommand\babelfont{\bbl@error{only-lua-xe}}{}{}{}
2148 \providecommand\babelprehyphenation{\bbl@error{only-lua}}{}{}{}
2149 \ifx\babelposthyphenation\@undefined
2150 \let\babelposthyphenation\babelprehyphenation
2151 \let\babelpatterns\babelprehyphenation
2152 \let\babelcharproperty\babelprehyphenation
2153 \fi
2154 </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.

```

2155 < *package >
2156 \bbl@trace{Creating languages and reading ini files}
2157 \let\bbl@extend@ini\gobble
2158 \newcommand\babelprovide[2][]{%
2159 \let\bbl@savelangname\languagename
2160 \edef\bbl@savelocaleid{\the\localeid}%
2161 % Set name and locale id
2162 \edef\languagename{#2}%
2163 \bbl@id@assign
2164 % Initialize keys

```

```

2165 \bbl@vforeach{captions,date,import,main,script,language,%
2166     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2167     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2168     Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2169 {\bbl@csarg\let{KVP@##1}\@nnil}%
2170 \global\let\bbl@release@transforms\@empty
2171 \global\let\bbl@release@casing\@empty
2172 \let\bbl@calendars\@empty
2173 \global\let\bbl@inidata\@empty
2174 \global\let\bbl@extend@ini\@gobble
2175 \global\let\bbl@included@inis\@empty
2176 \gdef\bbl@key@list{;}%
2177 \bbl@ifunset{bbl@passto@#2}%
2178     {\def\bbl@tempa{#1}}%
2179     {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}%
2180 \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2181     \in@{/}{#1}% With /, (re)sets a value in the ini
2182     \ifin@
2183         \global\let\bbl@extend@ini\bbl@extend@ini@aux
2184         \bbl@renewinikey##1\@{#2}%
2185     \else
2186         \bbl@csarg\ifx{KVP@##1}\@nnil\else
2187             \bbl@error{unknown-provide-key}{#1}{}%
2188         \fi
2189         \bbl@csarg\def{KVP@##1}{#2}%
2190     \fi}%
2191 \chardef\bbl@howloaded=0:none;1:ldf without ini;2:ini
2192 \bbl@ifunset{date#2}\z{\bbl@ifunset{bbl@llevel@#2}\one\tw@}%
2193 % == init ==
2194 \ifx\bbl@screset\undefined
2195     \bbl@ldfinit
2196 \fi
2197 % ==
2198 \ifx\bbl@KVP@import\@nnil\else \ifx\bbl@KVP@import\@nnil
2199     \def\bbl@KVP@import{\@empty}%
2200 \fi\fi
2201 % == date (as option) ==
2202 % \ifx\bbl@KVP@date\@nnil\else
2203 % \fi
2204 % ==
2205 \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2206 \ifcase\bbl@howloaded
2207     \let\bbl@lbkflag\@empty % new
2208 \else
2209     \ifx\bbl@KVP@hyphenrules\@nnil\else
2210         \let\bbl@lbkflag\@empty
2211     \fi
2212     \ifx\bbl@KVP@import\@nnil\else
2213         \let\bbl@lbkflag\@empty
2214     \fi
2215 \fi
2216 % == import, captions ==
2217 \ifx\bbl@KVP@import\@nnil\else
2218     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2219     {\ifx\bbl@initoload\relax
2220         \begingroup
2221             \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2222             \bbl@input@texini{#2}%
2223         \endgroup
2224     \else
2225         \xdef\bbl@KVP@import{\bbl@initoload}%
2226     \fi}%
2227 {}%

```

```

2228 \let\bbl@KVP@date\@empty
2229 \fi
2230 \let\bbl@KVP@captions@\bbl@KVP@captions
2231 \ifx\bbl@KVP@captions\@nnil
2232 \let\bbl@KVP@captions\bbl@KVP@import
2233 \fi
2234 % ==
2235 \ifx\bbl@KVP@transforms\@nnil\else
2236 \bbl@replace\bbl@KVP@transforms{ }{,}%
2237 \fi
2238 % == Load ini ==
2239 \ifcase\bbl@howloaded
2240 \bbl@provide@new{#2}%
2241 \else
2242 \bbl@ifblank{#1}%
2243 {}% With \bbl@load@basic below
2244 {\bbl@provide@renew{#2}}%
2245 \fi
2246 % == include == TODO
2247 % \ifx\bbl@included@inis\@empty\else
2248 % \bbl@replace\bbl@included@inis{ }{,}%
2249 % \bbl@foreach\bbl@included@inis{%
2250 % \openin\bbl@readstream=babel-##1.ini
2251 % \bbl@extend@ini{#2}}%
2252 % \closein\bbl@readstream
2253 % \fi
2254 % Post tasks
2255 % -----
2256 % == subsequent calls after the first provide for a locale ==
2257 \ifx\bbl@inidata\@empty\else
2258 \bbl@extend@ini{#2}%
2259 \fi
2260 % == ensure captions ==
2261 \ifx\bbl@KVP@captions\@nnil\else
2262 \bbl@ifunset{bbl@extracaps@#2}%
2263 {\bbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2264 {\bbl@exp{\\babelensure[exclude=\\today,
2265 include=\\bbl@extracaps@#2]}{#2}}%
2266 \bbl@ifunset{bbl@ensure@\\language}%
2267 {\bbl@exp{%
2268 \\DeclareRobustCommand<bbl@ensure@\\language>[1]{%
2269 \\foreignlanguage{\\language}%
2270 {###1}}}%
2271 }%
2272 \bbl@exp{%
2273 \\bbl@tglobal<bbl@ensure@\\language>%
2274 \\bbl@tglobal<bbl@ensure@\\language\space>%
2275 \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.

```

2276 \bbl@load@basic{#2}%
2277 % == script, language ==
2278 % Override the values from ini or defines them
2279 \ifx\bbl@KVP@script\@nnil\else
2280 \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2281 \fi
2282 \ifx\bbl@KVP@language\@nnil\else
2283 \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2284 \fi
2285 \ifcase\bbl@engine\or
2286 \bbl@ifunset{bbl@chrng@\\language}{}%

```



```

2287     {\directlua{
2288       Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2289 \fi
2290 % == Line breaking: intraspace, intrapenalty ==
2291 % For CJK, East Asian, Southeast Asian, if interspace in ini
2292 \ifx\bbl@KVP@intraspace@nnil\else % We can override the ini or set
2293   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2294 \fi
2295 \bbl@provide@intraspace
2296 % == Line breaking: justification ==
2297 \ifx\bbl@KVP@justification@nnil\else
2298   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2299 \fi
2300 \ifx\bbl@KVP@linebreaking@nnil\else
2301   \bbl@xin@{,\bbl@KVP@linebreaking,}%
2302   {,elongated,kashida,cjk,padding,unhyphenated,}%
2303   \ifin@
2304     \bbl@csarg\xdef
2305       {lnbrk@language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2306   \fi
2307 \fi
2308 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2309 \ifin@else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2310 \ifin@\bbl@arabicjust\fi
2311 % WIP
2312 \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2313 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2314 % == Line breaking: hyphenate.other.(locale|script) ==
2315 \ifx\bbl@lbkflag@empty
2316   \bbl@ifunset{bbl@hyotl@language}{}%
2317   {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
2318     \bbl@startcommands*{language}{}%
2319     \bbl@csarg\bbl@foreach{hyotl@language}{%
2320       \ifcase\bbl@engine
2321         \ifnum##1<257
2322           \SetHyphenMap{\BabelLower{##1}{##1}}%
2323         \fi
2324       \else
2325         \SetHyphenMap{\BabelLower{##1}{##1}}%
2326       \fi}%
2327     \bbl@endcommands}%
2328   \bbl@ifunset{bbl@hyots@language}{}%
2329   {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2330     \bbl@csarg\bbl@foreach{hyots@language}{%
2331       \ifcase\bbl@engine
2332         \ifnum##1<257
2333           \global\lccode##1=##1\relax
2334         \fi
2335       \else
2336         \global\lccode##1=##1\relax
2337       \fi}%
2338   \fi
2339 % == Counters: maparabic ==
2340 % Native digits, if provided in ini (TeX level, xe and lua)
2341 \ifcase\bbl@engine\else
2342   \bbl@ifunset{bbl@dgnat@language}{}%
2343   {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
2344     \expandafter\expandafter\expandafter
2345     \bbl@setdigits\csname bbl@dgnat@language\endcsname
2346     \ifx\bbl@KVP@maparabic@nnil\else
2347       \ifx\bbl@latin@arabic@undefined
2348         \expandafter\let\expandafter\@arabic
2349         \csname bbl@counter@language\endcsname

```

```

2350         \else      % i.e., if layout=counters, which redefines \@arabic
2351             \expandafter\let\expandafter\bbl@latinarabic
2352             \csname bbl@counter@\language\endcsname
2353         \fi
2354     \fi
2355 \fi}%
2356 \fi
2357 % == Counters: mapdigits ==
2358 % > luababel.def
2359 % == Counters: alph, Alph ==
2360 \ifx\bbl@KVP@alph\@nnil\else
2361     \bbl@exp{%
2362         \\bbl@add\<bbl@preextras@\language\>{%
2363             \\bbl@save\\@alph
2364             \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\language\>}}%
2365 \fi
2366 \ifx\bbl@KVP@Alph\@nnil\else
2367     \bbl@exp{%
2368         \\bbl@add\<bbl@preextras@\language\>{%
2369             \\bbl@save\\@Alph
2370             \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\language\>}}%
2371 \fi
2372 % == Casing ==
2373 \bbl@release@casing
2374 \ifx\bbl@KVP@casing\@nnil\else
2375     \bbl@csarg\xdef{casing@\language}%
2376     {\@nameuse{bbl@casing@\language}}\bbl@maybextx\bbl@KVP@casing}%
2377 \fi
2378 % == Calendars ==
2379 \ifx\bbl@KVP@calendar\@nnil
2380     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2381 \fi
2382 \def\bbl@tempe##1 ##2\@{ % Get first calendar
2383     \def\bbl@tempa{##1}}%
2384     \bbl@exp{\\bbl@tempe\bbl@KVP@calendar\space\\@}%
2385 \def\bbl@tempe##1.##2.##3\@{ %
2386     \def\bbl@tempc{##1}%
2387     \def\bbl@tempb{##2}}%
2388 \expandafter\bbl@tempe\bbl@tempa.\@
2389 \bbl@csarg\edef{calpr@\language}{%
2390     \ifx\bbl@tempc\@empty\else
2391         calendar=\bbl@tempc
2392     \fi
2393     \ifx\bbl@tempb\@empty\else
2394         ,variant=\bbl@tempb
2395     \fi}%
2396 % == engine specific extensions ==
2397 % Defined in XXXbabel.def
2398 \bbl@provide@extra{#2}%
2399 % == require.babel in ini ==
2400 % To load or reload the babel-*.tex, if require.babel in ini
2401 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2402     \bbl@ifunset{bbl@rqtex@\language}{}%
2403     {\expandafter\ifx\csname bbl@rqtex@\language\endcsname\@empty\else
2404         \let\BabelBeforeIni\@gobbletwo
2405         \chardef\atcatcode=\catcode\@
2406         \catcode\@=11\relax
2407         \def\CurrentOption{#2}%
2408         \bbl@input{texini{\bbl@cs{rqtex@\language}}}%
2409         \catcode\@=\atcatcode
2410         \let\atcatcode\relax
2411         \global\bbl@csarg\let{rqtex@\language}\relax
2412     \fi}%

```

```

2413 \bbl@foreach\bbl@calendars{%
2414 \bbl@ifunset\bbl@ca@##1}{%
2415 \chardef\atcatcode=\catcode\@
2416 \catcode\@=11\relax
2417 \InputIfFileExists{babel-ca-##1.tex}{}}{%
2418 \catcode\@=\atcatcode
2419 \let\atcatcode\relax}%
2420 }}%
2421 \fi
2422 % == frenchspacing ==
2423 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2424 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2425 \ifin@
2426 \bbl@extras@wrap{\bbl@pre@fs}%
2427 {\bbl@pre@fs}%
2428 {\bbl@post@fs}%
2429 \fi
2430 % == transforms ==
2431 % > luababel.def
2432 \def\CurrentOption{#2}%
2433 \@nameuse{\bbl@icsave@#2}%
2434 % == main ==
2435 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2436 \let\language\bbl@savelangname
2437 \chardef\localeid\bbl@savelocaleid\relax
2438 \fi
2439 % == hyphenrules (apply if current) ==
2440 \ifx\bbl@KVP@hyphenrules\@nnil\else
2441 \ifnum\bbl@savelocaleid=\localeid
2442 \language\@nameuse{l\language}%
2443 \fi
2444 \fi}

```

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

```

2445 \def\bbl@provide@new#1{%
2446 \namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2447 \namedef{extras#1}{}%
2448 \namedef{noextras#1}{}%
2449 \bbl@startcommands*{#1}{captions}%
2450 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2451 \def\bbl@tempb##1{% elt for \bbl@captionslist
2452 \ifx##1\@nnil\else
2453 \bbl@exp{%
2454 \SetString\##1{%
2455 \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2456 \expandafter\bbl@tempb
2457 \fi}%
2458 \expandafter\bbl@tempb\bbl@captionslist\@nnil
2459 \else
2460 \ifx\bbl@initoload\relax
2461 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2462 \else
2463 \bbl@read@ini{\bbl@initoload}2% % Same
2464 \fi
2465 \fi
2466 \StartBabelCommands*{#1}{date}%
2467 \ifx\bbl@KVP@date\@nnil
2468 \bbl@exp{%
2469 \SetString\today{\bbl@nocaption{today}{#1today}}}%
2470 \else
2471 \bbl@savetoday
2472 \bbl@savedate

```

```

2473 \fi
2474 \bbl@endcommands
2475 \bbl@load@basic{#1}%
2476 % == hyphenmins == (only if new)
2477 \bbl@exp{%
2478 \gdef\<#1hyphenmins>{%
2479 {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2480 {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
2481 % == hyphenrules (also in renew) ==
2482 \bbl@provide@hyphens{#1}%
2483 \ifx\bbl@KVP@main\@nnil\else
2484 \expandafter\main@language\expandafter{#1}%
2485 \fi}
2486 %
2487 \def\bbl@provide@renew#1{%
2488 \ifx\bbl@KVP@captions\@nnil\else
2489 \StartBabelCommands*{#1}{captions}%
2490 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2491 \EndBabelCommands
2492 \fi
2493 \ifx\bbl@KVP@date\@nnil\else
2494 \StartBabelCommands*{#1}{date}%
2495 \bbl@savetoday
2496 \bbl@savedate
2497 \EndBabelCommands
2498 \fi
2499 % == hyphenrules (also in new) ==
2500 \ifx\bbl@lbkflag\@empty
2501 \bbl@provide@hyphens{#1}%
2502 \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.

```

2503 \def\bbl@load@basic#1{%
2504 \ifcase\bbl@howloaded\or\or
2505 \ifcase\csname bbl@llevel@\language\endcsname
2506 \bbl@csarg\let\lname@\language\relax
2507 \fi
2508 \fi
2509 \bbl@ifunset{\bbl@lname@#1}%
2510 {\def\BabelBeforeIni##1##2{%
2511 \begingroup
2512 \let\bbl@ini@captions@aux\@gobbletwo
2513 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2514 \bbl@read@ini{##1}1%
2515 \ifx\bbl@initoload\relax\endinput\fi
2516 \endgroup}%
2517 \begingroup % boxed, to avoid extra spaces:
2518 \ifx\bbl@initoload\relax
2519 \bbl@input@texini{#1}%
2520 \else
2521 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2522 \fi
2523 \endgroup}%
2524 {}}

```

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`.

```

2525 \def\bbl@provide@hyphens#1{%
2526 \@tempcnta\m@ne % a flag
2527 \ifx\bbl@KVP@hyphenrules\@nnil\else
2528 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2529 \bbl@foreach\bbl@KVP@hyphenrules{%

```

```

2530 \ifnum\@tempcnta=\m@ne % if not yet found
2531 \bbl@ifsamestring{##1}{+}%
2532 {\bbl@carg\addlanguage{l@##1}}%
2533 }%
2534 \bbl@ifunset{l@##1}% After a possible +
2535 }%
2536 {\@tempcnta\@nameuse{l@##1}}%
2537 \fi}%
2538 \ifnum\@tempcnta=\m@ne
2539 \bbl@warning{%
2540 Requested 'hyphenrules' for '\language' not found:\%
2541 \bbl@KVP@hyphenrules.\%
2542 Using the default value. Reported}%
2543 \fi
2544 \fi
2545 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2546 \ifx\bbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2547 \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2548 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2549 }%
2550 {\bbl@ifunset{l@bbl@cl{hyphr}}}%
2551 }% if hyphenrules found:
2552 {\@tempcnta\@nameuse{l@bbl@cl{hyphr}}}%
2553 \fi
2554 \fi
2555 \bbl@ifunset{l@#1}%
2556 {\ifnum\@tempcnta=\m@ne
2557 \bbl@carg\adddialect{l@#1}\language
2558 \else
2559 \bbl@carg\adddialect{l@#1}\@tempcnta
2560 \fi}%
2561 {\ifnum\@tempcnta=\m@ne\else
2562 \global\bbl@carg\chardef{l@#1}\@tempcnta
2563 \fi}}

```

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

```

2564 \def\bbl@input@texini#1{%
2565 \bbl@bsphack
2566 \bbl@exp{%
2567 \catcode`\\%=14 \catcode`\\%=0
2568 \catcode`\\={1 \catcode`\\}=2
2569 \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2570 \catcode`\\%=the\catcode`\%relax
2571 \catcode`\\%=the\catcode`\%relax
2572 \catcode`\\={the\catcode`\%relax
2573 \catcode`\\}=the\catcode`\%relax}%
2574 \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.

```

2575 \def\bbl@inline#1\bbl@inline{%
2576 \ifnextchar[\bbl@iniset{\ifnextchar\bbl@iniskip\bbl@inistore}#1\@@% ]
2577 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2578 \def\bbl@iniskip#1\@@{% if starts with ;
2579 \def\bbl@inistore#1=#2\@@{% full (default)
2580 \bbl@trim@def\bbl@tempa{#1}%
2581 \bbl@trim\toks{#2}%
2582 \bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2583 \ifin@else
2584 \bbl@xin@{,identification/include.}%
2585 {\bbl@section/\bbl@tempa}%
2586 \ifin@\xdef\bbl@included@inis{the\toks@}\fi

```

```

2587 \bbl@exp{%
2588   \\g@addto@macro\\bbl@inidata{%
2589     \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2590 \fi}
2591 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2592   \bbl@trim@def\bbl@tempa{#1}%
2593   \bbl@trim\toks@{#2}%
2594   \bbl@xin@{.identification.}{.\bbl@section.}%
2595   \ifin@
2596     \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2597       \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2598   \fi}

```

4.19. Main loop in ‘provide’

Now, the ‘main loop’, 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 or 2.

```

2599 \def\bbl@loop@ini{%
2600   \loop
2601     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2602     \endlinechar\m@ne
2603     \read\bbl@readstream to \bbl@line
2604     \endlinechar`\^^M
2605     \ifx\bbl@line\empty\else
2606       \expandafter\bbl@iniline\bbl@line\bbl@iniline
2607     \fi
2608   \repeat}
2609 \ifx\bbl@readstream\@undefined
2610   \csname newread\endcsname\bbl@readstream
2611 \fi
2612 \def\bbl@read@ini#1#2{%
2613   \global\let\bbl@extend@ini@gobble
2614   \openin\bbl@readstream=babel-#1.ini
2615   \ifeof\bbl@readstream
2616     \bbl@error{no-ini-file}{#1}{}}}%
2617 \else
2618   % == Store ini data in \bbl@inidata ==
2619   \catcode`\[=12 \catcode`\]=12 \catcode`\&=12 \catcode`\&=12
2620   \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2621   \bbl@info{Importing
2622     \ifcase#2font and identification \or basic \fi
2623     data for \language\name\\%
2624     from babel-#1.ini. Reported}%
2625   \ifnum#2=\z@
2626     \global\let\bbl@inidata\@empty
2627     \let\bbl@inistore\bbl@inistore@min % Remember it's local
2628   \fi
2629   \def\bbl@section{identification}%
2630   \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2631   \bbl@inistore load.level=#2\@@
2632   \bbl@loop@ini
2633   % == Process stored data ==
2634   \bbl@csarg\xdef{lini@\language\name}{#1}%
2635   \bbl@read@ini@aux
2636   % == 'Export' data ==
2637   \bbl@ini@exports{#2}%
2638   \global\bbl@csarg\let{inidata@\language\name}\bbl@inidata
2639   \global\let\bbl@inidata\@empty
2640   \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language\name}}}%

```

```

2641 \bbl@toglobal\bbl@ini@loaded
2642 \fi
2643 \closein\bbl@readstream}
2644 \def\bbl@read@ini@aux{%
2645 \let\bbl@savestrings\@empty
2646 \let\bbl@savetoday\@empty
2647 \let\bbl@savestate\@empty
2648 \def\bbl@elt##1##2##3{%
2649 \def\bbl@section{##1}%
2650 \in@{=date.}{=##1}% Find a better place
2651 \ifin@
2652 \bbl@ifunset{bbl@inikv@##1}%
2653 {\bbl@ini@calendar{##1}}%
2654 {}%
2655 \fi
2656 \bbl@ifunset{bbl@inikv@##1}{}%
2657 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2658 \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.

```

2659 \def\bbl@extend@ini@aux#1{%
2660 \bbl@startcommands*{#1}{captions}%
2661 % Activate captions/... and modify exports
2662 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2663 \setlocalecaption{#1}{##1}{##2}}%
2664 \def\bbl@inikv@captions##1##2{%
2665 \bbl@ini@captions@aux{##1}{##2}}%
2666 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2667 \def\bbl@exportkey##1##2##3{%
2668 \bbl@ifunset{bbl@kv@##2}{}%
2669 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2670 \bbl@exp{\global\let<bbl@##1@\language>\<bbl@kv@##2>}}%
2671 \fi}}%
2672 % As with \bbl@read@ini, but with some changes
2673 \bbl@read@ini@aux
2674 \bbl@ini@exports\tw@
2675 % Update inidata@lang by pretending the ini is read.
2676 \def\bbl@elt##1##2##3{%
2677 \def\bbl@section{##1}%
2678 \bbl@iniline##2=##3\bbl@iniline}%
2679 \csname bbl@inidata@#1\endcsname
2680 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2681 \StartBabelCommands*{#1}{date}% And from the import stuff
2682 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2683 \bbl@savetoday
2684 \bbl@savestate
2685 \bbl@endcommands}

```

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

```

2686 \def\bbl@ini@calendar#1{%
2687 \lowercase{\def\bbl@tempa{=##1=}}%
2688 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2689 \bbl@replace\bbl@tempa{=date.}{}%
2690 \in@{.licr=}{#1=}%
2691 \ifin@
2692 \ifcase\bbl@engine
2693 \bbl@replace\bbl@tempa{.licr=}{}%
2694 \else
2695 \let\bbl@tempa\relax
2696 \fi
2697 \fi
2698 \ifx\bbl@tempa\relax\else
2699 \bbl@replace\bbl@tempa{=}{}%

```

```

2700 \ifx\bbbl@tempa\@empty\else
2701 \xdef\bbbl@calendars{\bbbl@calendars,\bbbl@tempa}%
2702 \fi
2703 \bbbl@exp{%
2704 \def<\bbbl@inikv@#1>####1####2{%
2705 \\\bbbl@inidate####1...\relax{####2}{\bbbl@tempa}}}%
2706 \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 \bbbl@inistore above).

```

2707 \def\bbbl@renewinikey#1/#2\@#3{%
2708 \edef\bbbl@tempa{\zap@space #1 \@empty}% section
2709 \edef\bbbl@tempb{\zap@space #2 \@empty}% key
2710 \bbbl@trim\toks@{#3}% value
2711 \bbbl@exp{%
2712 \edef\\bbbl@key@list{\bbbl@key@list \bbbl@tempa/\bbbl@tempb;}%
2713 \\g@addto@macro\\bbbl@inidata{%
2714 \\\bbbl@elt{\bbbl@tempa}{\bbbl@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.

```

2715 \def\bbbl@exportkey#1#2#3{%
2716 \bbbl@ifunset{\bbbl@kv@#2}%
2717 {\bbbl@csarg\gdef{#1@\language\language}{#3}}%
2718 {\expandafter\ifx\csname \bbbl@kv@#2\endcsname\@empty
2719 \bbbl@csarg\gdef{#1@\language\language}{#3}%
2720 \else
2721 \bbbl@exp{\global\let<\bbbl@#1@\language\language>\<\bbbl@kv@#2>}%
2722 \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 \bbbl@ini@exports is called always (via \bbbl@inisec), while \bbbl@after@ini must be called explicitly after \bbbl@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.

```

2723 \def\bbbl@iniwarning#1{%
2724 \bbbl@ifunset{\bbbl@kv@identification.warning#1}{}%
2725 {\bbbl@warning{%
2726 From babel-\bbbl@cs{lini@\language\language}.ini:\\%
2727 \bbbl@cs{@kv@identification.warning#1}\\%
2728 Reported }}}
2729 %
2730 \let\bbbl@release@transforms\@empty
2731 \let\bbbl@release@casing\@empty
2732 \def\bbbl@ini@exports#1{%
2733 % Identification always exported
2734 \bbbl@iniwarning{}%
2735 \ifcase\bbbl@engine
2736 \bbbl@iniwarning{.pdf\latex}%
2737 \or
2738 \bbbl@iniwarning{.lua\latex}%
2739 \or
2740 \bbbl@iniwarning{.xel\latex}%
2741 \fi%
2742 \bbbl@exportkey{lllevel}{identification.load.level}{}%
2743 \bbbl@exportkey{elname}{identification.name.english}{}%
2744 \bbbl@exp{\\bbbl@exportkey{lname}{identification.name.opentype}%
2745 {\csname \bbbl@elname@\language\language\endcsname}}%
2746 \bbbl@exportkey{tbc}{identification.tag.bcp47}{}%
2747 % Somewhat hackish. TODO:

```



```

2748 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2749 \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2750 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2751 \bbl@exportkey{esname}{identification.script.name}{}%
2752 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2753 { \csname bbl@esname@ \language \endcsname }}%
2754 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2755 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2756 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2757 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2758 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2759 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2760 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2761 % Also maps bcp47 -> language
2762 \ifbbl@bcptoname
2763 \bbl@csarg\xdef{bcp@map@ \bbl@cl{tbc}}{\language}%
2764 \fi
2765 \ifcase\bbl@engine\or
2766 \directlua{%
2767     Babel.locale_props[\the\bbl@cs{id@ \language}].script
2768     = '\bbl@cl{sbc}}'%
2769 \fi
2770 % Conditional
2771 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
2772 \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2773 \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
2774 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2775 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2776 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2777 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2778 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2779 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2780 \bbl@exportkey{intsp}{typography.intraspace}{}%
2781 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2782 \bbl@exportkey{chrng}{characters.ranges}{}%
2783 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2784 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2785 \ifnum#1=\tw@ % only (re)new
2786 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2787 \bbl@tglobal\bbl@savetoday
2788 \bbl@tglobal\bbl@savestate
2789 \bbl@savestrings
2790 \fi
2791 \fi}

```

4.20. Processing keys in ini

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

```

2792 \def\bbl@inikv#1#2{%      key=value
2793 \toks@{#2}%               This hides #'s from ini values
2794 \bbl@csarg\xdef{kv@ \bbl@section.#1}{\the\toks@}}

```

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

```

2795 \let\bbl@inikv@identification\bbl@inikv
2796 \let\bbl@inikv@date\bbl@inikv
2797 \let\bbl@inikv@typography\bbl@inikv
2798 \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`.

```

2799 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@ \language}\@empty x-\fi}
2800 \def\bbl@inikv@characters#1#2{%

```

```

2801 \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2802 {\bbl@exp{%
2803     \\g@addto@macro\\bbl@release@casing{%
2804         \\bbl@casemapping{\\language\\}{\\unexpanded{#2}}}%
2805     {\in@{casing.}{#1}% e.g., casing.Uv = uV
2806         \ifin@
2807             \lowercase{\def\bbl@tempb{#1}}%
2808             \bbl@replace\bbl@tempb{casing.}{}%
2809             \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2810                 \\bbl@casemapping
2811                 {\\bbl@maybextx\bbl@tempb}{\\language\\}{\\unexpanded{#2}}}%
2812             \else
2813                 \bbl@inikv{#1}{#2}%
2814             \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’.

```

2815 \def\bbl@inikv@counters#1#2{%
2816     \bbl@ifsamestring{#1}{digits}%
2817     {\bbl@error{digits-is-reserved}{}}}%
2818     {}%
2819 \def\bbl@tempc{#1}%
2820 \bbl@trim@def{\bbl@tempb*}{#2}%
2821 \in@{.1}{#1}%
2822 \ifin@
2823     \bbl@replace\bbl@tempc{.1}{}%
2824     \bbl@csarg\protected@xdef{cnt@#1\bbl@tempc @\language}{%
2825         \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2826 \fi
2827 \in@{.F.}{#1}%
2828 \ifin@else\in@{.S.}{#1}\fi
2829 \ifin@
2830     \bbl@csarg\protected@xdef{cnt@#1@\language}{\bbl@tempb*}%
2831 \else
2832     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2833     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2834     \bbl@csarg{\global\expandafter\let}{cnt@#1@\language}\bbl@tempa
2835 \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.

```

2836 \ifcase\bbl@engine
2837     \bbl@csarg\def{inikv@captions.licr}#1#2{%
2838         \bbl@ini@captions@aux{#1}{#2}}
2839 \else
2840     \def\bbl@inikv@captions#1#2{%
2841         \bbl@ini@captions@aux{#1}{#2}}
2842 \fi

```

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

```

2843 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2844     \bbl@replace\bbl@tempa{.template}{}%
2845     \def\bbl@toreplace{#1}{}%
2846     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2847     \bbl@replace\bbl@toreplace{[ ]}{\csname}%
2848     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2849     \bbl@replace\bbl@toreplace{[ ]}{name\endcsname{}}%
2850     \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
2851     \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2852     \ifin@
2853         \@nameuse{\bbl@patch\bbl@tempa}%
2854     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace

```

```

2855 \fi
2856 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2857 \ifin@
2858 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2859 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2860 \\\bbl@ifunset{\bbl@tempa fmt@\\language}%
2861 \{fnum@\bbl@tempa}}%
2862 \{\@nameuse{\bbl@tempa fmt@\\language}}}%
2863 \fi}
2864 \def\bbl@ini@captions@aux#1#2{%
2865 \bbl@trim@def\bbl@tempa{#1}%
2866 \bbl@xin@{.template}{\bbl@tempa}%
2867 \ifin@
2868 \bbl@ini@captions@template{#2}\language
2869 \else
2870 \bbl@ifblank{#2}%
2871 {\bbl@exp{%
2872 \toks@{\\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2873 {\bbl@trim\toks@{#2}}}%
2874 \bbl@exp{%
2875 \\\bbl@add\\bbl@savestrings{%
2876 \\\SetString\<\bbl@tempa name>{\the\toks@}}%
2877 \toks@expandafter{\bbl@captionslist}%
2878 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
2879 \ifin@else
2880 \bbl@exp{%
2881 \\\bbl@add\<\bbl@extracaps@language>{\<\bbl@tempa name>%
2882 \\\bbl@toglobal\<\bbl@extracaps@language>}}%
2883 \fi
2884 \fi}

```

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

```

2885 \def\bbl@list@the{%
2886 part,chapter,section,subsection,subsubsection,paragraph,%
2887 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2888 table,page,footnote,mpfootnote,mpfn}
2889 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2890 \bbl@ifunset{\bbl@map@#1@language}%
2891 {\@nameuse{#1}}%
2892 {\@nameuse{\bbl@map@#1@language}}}
2893 \def\bbl@inikv@labels#1#2{%
2894 \in@{.map}{#1}%
2895 \ifin@
2896 \ifx\bbl@KVP@labels\@nnil\else
2897 \bbl@xin@{ map }{\bbl@KVP@labels\space}%
2898 \ifin@
2899 \def\bbl@tempc{#1}%
2900 \bbl@replace\bbl@tempc{.map}{}%
2901 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
2902 \bbl@exp{%
2903 \gdef\<\bbl@map@\bbl@tempc @language>%
2904 {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
2905 \bbl@foreach\bbl@list@the{%
2906 \bbl@ifunset{the##1}{}%
2907 {\bbl@exp{\let\\bbl@tempd\<the##1>}%
2908 \bbl@exp{%
2909 \\\bbl@sreplace\<the##1>%
2910 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
2911 \\\bbl@sreplace\<the##1>%
2912 {\<\empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2913 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2914 \toks@\expandafter\expandafter\expandafter{%
2915 \csname the##1\endcsname}%

```

```

2916         \expandafter\edef\csname the##1\endcsname{\the\toks@}%
2917     \fi}%
2918     \fi
2919     \fi
2920     %
2921     \else
2922     %
2923     % The following code is still under study. You can test it and make
2924     % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
2925     % language dependent.
2926     \in@{enumerate.}{#1}%
2927     \ifin@
2928         \def\bbl@tempa{#1}%
2929         \bbl@replace\bbl@tempa{enumerate.}{}%
2930         \def\bbl@toreplace{#2}%
2931         \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2932         \bbl@replace\bbl@toreplace{[]}{\csname the}%
2933         \bbl@replace\bbl@toreplace{[]}{\endcsname{}}}%
2934         \toks@\expandafter{\bbl@toreplace}%
2935         % TODO. Execute only once:
2936         \bbl@exp{%
2937             \\bbl@add<extras\language>{%
2938                 \\babel@save<labelenum\romannumeral\bbl@tempa>%
2939                 \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
2940             \\bbl@tglobal<extras\language>}%
2941     \fi
2942     \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.

```

2943 \def\bbl@chapttype{chapter}
2944 \ifx\@makechapterhead\undefined
2945     \let\bbl@patchchapter\relax
2946 \else\ifx\thechapter\undefined
2947     \let\bbl@patchchapter\relax
2948 \else\ifx\ps@headings\undefined
2949     \let\bbl@patchchapter\relax
2950 \else
2951     \def\bbl@patchchapter{%
2952         \global\let\bbl@patchchapter\relax
2953         \gdef\bbl@chfmt{%
2954             \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
2955             {\@chapapp\space\thechapter}{\bbl@chfmt}%
2956             {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}%
2957         \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
2958         \bbl@sreplace\ps@headings{\@chapapp\space\thechapter}{\bbl@chfmt}%
2959         \bbl@sreplace\chaptermark{\@chapapp\space\thechapter}{\bbl@chfmt}%
2960         \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
2961         \bbl@tglobal\appendix
2962         \bbl@tglobal\ps@headings
2963         \bbl@tglobal\chaptermark
2964         \bbl@tglobal\@makechapterhead}
2965     \let\bbl@patchappendix\bbl@patchchapter
2966 \fi\fi\fi
2967 \ifx\@part\undefined
2968     \let\bbl@patchpart\relax
2969 \else
2970     \def\bbl@patchpart{%
2971         \global\let\bbl@patchpart\relax
2972         \gdef\bbl@partformat{%
2973             \bbl@ifunset{\bbl@partfmt@\language}%

```

```

2974         {\partname\nobreakspace\thepart}
2975         {\@nameuse{bbl@partfmt@\language\language}}
2976         \bbl@replace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
2977         \bbl@tglobal\@part}
2978 \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

```

2979 \let\bbl@calendar\@empty
2980 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
2981 \def\bbl@localedate#1#2#3#4{%
2982   \begingroup
2983     \edef\bbl@they{#2}%
2984     \edef\bbl@them{#3}%
2985     \edef\bbl@thed{#4}%
2986     \edef\bbl@tempe{%
2987       \bbl@ifunset{bbl@calpr@\language\language}{\bbl@cl{calpr}},%
2988       #1}%
2989     \bbl@replace\bbl@tempe{ }{}%
2990     \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
2991     \bbl@replace\bbl@tempe{convert}{convert=}%
2992     \let\bbl@ld@calendar\@empty
2993     \let\bbl@ld@variant\@empty
2994     \let\bbl@ld@convert\relax
2995     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld##1}{##2}}%
2996     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
2997     \bbl@replace\bbl@ld@calendar{gregorian}{}%
2998     \ifx\bbl@ld@calendar\@empty\else
2999       \ifx\bbl@ld@convert\relax\else
3000         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3001         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3002       \fi
3003     \fi
3004     \@nameuse{bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3005     \edef\bbl@calendar{% Used in \month..., too
3006       \bbl@ld@calendar
3007       \ifx\bbl@ld@variant\@empty\else
3008         .\bbl@ld@variant
3009       \fi}%
3010     \bbl@cased
3011     {\@nameuse{bbl@date@\language\language @\bbl@calendar}%
3012      \bbl@they\bbl@them\bbl@thed}%
3013   \endgroup}
3014 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3015 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3016   \bbl@trim@def\bbl@tempa{#1.#2}%
3017   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3018   {\bbl@trim@def\bbl@tempa{#3}%
3019    \bbl@trim\toks@{#5}%
3020    \@temptokena\expandafter{\bbl@savestate}%
3021    \bbl@exp{% Reverse order - in ini last wins
3022      \def\\bbl@savestate{%
3023        \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3024        \the\@temptokena}}}%
3025   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3026    {\lowercase{\def\bbl@tempb{#6}}}%
3027    \bbl@trim@def\bbl@toreplace{#5}%
3028    \bbl@TG@@date
3029    \global\bbl@csarg\let{date@\language\language @\bbl@tempb}\bbl@toreplace
3030    \ifx\bbl@savetoday\@empty
3031      \bbl@exp{% TODO. Move to a better place.
3032        \\AfterBabelCommands{%
3033          \gdef\<\language\language date>{\protect\<\language\language date >}}%

```

```

3034         \gdef\<\language name date >{\bbl@printdate{\language name}}}%
3035     \def\bbl@savetoday{%
3036         \SetString\today{%
3037             \<\language name date>[convert]%
3038             {\the\year}{\the\month}{\the\day}}}%
3039     \fi}%
3040 }}}}
3041 \def\bbl@printdate#1{%
3042     \ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}}
3043 \def\bbl@printdate@i#1[#2]#3#4#5{%
3044     \bbl@usedategrouptrue
3045     \@nameuse{bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}

```

4.21. French spacing (again)

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

```

3046 \AddToHook{begindocument/before}{%
3047     \let\bbl@normalsf\normalsfcodes
3048     \let\normalsfcodes\relax}
3049 \AtBeginDocument{%
3050     \ifx\bbl@normalsf\@empty
3051         \ifnum\sfcodes\@m
3052             \let\normalsfcodes\frenchspacing
3053         \else
3054             \let\normalsfcodes\nonfrenchspacing
3055         \fi
3056     \else
3057         \let\normalsfcodes\bbl@normalsf
3058     \fi}

```

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).

```

3059 \let\bbl@calendar\@empty
3060 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3061     \@nameuse{bbl@ca@#2}#1@@}
3062 \newcommand\babelDateSpace{\nobreakspace}
3063 \newcommand\babelDateDot{.\@} % TODO. \let instead of repeating
3064 \newcommand\babelDated[1]{\number#1}
3065 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3066 \newcommand\babelDateM[1]{\number#1}
3067 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3068 \newcommand\babelDateMMM[1]{%
3069     \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3070 \newcommand\babelDatey[1]{\number#1}%
3071 \newcommand\babelDateyy[1]{%
3072     \ifnum#1<10 0\number#1 %
3073     \else\ifnum#1<100 \number#1 %
3074     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3075     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3076     \else
3077         \bbl@error{limit-two-digits}{}}}%
3078     \fi\fi\fi\fi}
3079 \newcommand\babelDateyyyy[1]{\number#1} % TODO - add leading 0
3080 \newcommand\babelDateU[1]{\number#1}%
3081 \def\bbl@replace@finish@iii#1{%
3082     \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3083 \def\bbl@TG@date{%
3084     \bbl@replace\bbl@toreplace[ ]{\babelDateSpace}}%
3085     \bbl@replace\bbl@toreplace[.]{\babelDateDot}}%

```

```

3086 \bbl@replace\bbl@toreplace{[d]}\BabelDated{###3}%
3087 \bbl@replace\bbl@toreplace{[dd]}\BabelDateddd{###3}%
3088 \bbl@replace\bbl@toreplace{[M]}\BabelDateM{###2}%
3089 \bbl@replace\bbl@toreplace{[MM]}\BabelDateMM{###2}%
3090 \bbl@replace\bbl@toreplace{[MMMM]}\BabelDateMMMM{###2}%
3091 \bbl@replace\bbl@toreplace{[y]}\BabelDatey{###1}%
3092 \bbl@replace\bbl@toreplace{[yy]}\BabelDateyy{###1}%
3093 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{###1}%
3094 \bbl@replace\bbl@toreplace{[U]}\BabelDateU{###1}%
3095 \bbl@replace\bbl@toreplace{[y]}\bbl@datecctr{###1}%
3096 \bbl@replace\bbl@toreplace{[U]}\bbl@datecctr{###1}%
3097 \bbl@replace\bbl@toreplace{[m]}\bbl@datecctr{###2}%
3098 \bbl@replace\bbl@toreplace{[d]}\bbl@datecctr{###3}%
3099 \bbl@replace@finish@iii\bbl@toreplace}
3100 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3101 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3102 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3103 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3104 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3105   #1[#2]{#3}{#4}{#5}}
3106 \begingroup % A hack. TODO. Don't require a specific order
3107 \catcode\%=12
3108 \catcode\&=14
3109 \gdef\bbl@transforms#1#2#3{%&
3110   \directlua{
3111     local str = [=[#2]=]
3112     str = str:gsub('%.%d+%.%d+$', '')
3113     token.set_macro('babeltempa', str)
3114   }&
3115   \def\babeltempc{}&
3116   \bbl@xin@{\babeltempa,}{,\bbl@KVP@transforms,}&
3117   \ifin@else
3118     \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&
3119   \fi
3120   \ifin@
3121     \bbl@foreach\bbl@KVP@transforms{%&
3122       \bbl@xin@{: \babeltempa,}{,##1,}&
3123       \ifin@ & font:font:transform syntax
3124         \directlua{
3125           local t = {}
3126           for m in string.gmatch('##1'..' ':'', '(.):') do
3127             table.insert(t, m)
3128           end
3129           table.remove(t)
3130           token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3131         }&
3132       \fi}&
3133   \in@{.0$}{#2$}&
3134   \ifin@
3135     \directlua{% (\attribute) syntax
3136       local str = string.match([[ \bbl@KVP@transforms]],
3137         '%([^(%[-])^(%)]-\babeltempa)')
3138       if str == nil then
3139         token.set_macro('babeltempb', '')
3140       else
3141         token.set_macro('babeltempb', ',attribute=' .. str)
3142       end
3143     }&
3144   \toks@{#3}&
3145   \bbl@exp{%&
3146     \\g@addto@macro\\bbl@release@transforms{%&

```

```

3147         \relax &% Closes previous \bbl@transforms@aux
3148         \\bbl@transforms@aux
3149         \\\#1{label=\babeltempa\babeltempb\babeltempc}&%
3150         {\language\the\toks@}}&%
3151     \else
3152         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3153     \fi
3154 \fi}
3155 \endgroup

```

4.22. Handle language system

Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```

3156 \def\bbl@provide@lsys#1{%
3157   \bbl@ifunset{bbl@lname@#1}%
3158     {\bbl@load@info{#1}}%
3159   {%
3160     \bbl@csarg\let{lsys@#1}\empty
3161     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{%
3162       \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3163         \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3164         \bbl@ifunset{bbl@lname@#1}{%
3165           {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3166         \ifcase\bbl@engine\or\or
3167           \bbl@ifunset{bbl@prehc@#1}{%
3168             {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3169             {%
3170               {\ifx\bbl@xenohyph\undefined
3171                 \global\let\bbl@xenohyph\bbl@xenohyph@d
3172                 \ifx\AtBeginDocument\@notprerr
3173                   \expandafter\@secondoftwo % to execute right now
3174                 \fi
3175                 \AtBeginDocument{%
3176                   \bbl@patchfont{\bbl@xenohyph}%
3177                   {\expandafter\select@language\expandafter{\language}}}%
3178               \fi}}%
3179             \fi
3180             \bbl@csarg\bbl@toglobal{lsys@#1}}
3181   \def\bbl@xenohyph@d{%
3182     \bbl@ifset{bbl@prehc@\language}%
3183       {\ifnum\hyphenchar\font=\defaultthyphenchar
3184         \iffontchar\font\bbl@c{prehc}\relax
3185         \hyphenchar\font\bbl@c{prehc}\relax
3186       \else\iffontchar\font"200B
3187         \hyphenchar\font"200B
3188       \else
3189         \bbl@warning
3190         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3191         in the current font, and therefore the hyphen\\%
3192         will be printed. Try changing the fontspec's\\%
3193         'HyphenChar' to another value, but be aware\\%
3194         this setting is not safe (see the manual).\\%
3195         Reported}%
3196         \hyphenchar\font\defaultthyphenchar
3197       \fi\fi
3198     \fi}%
3199     {\hyphenchar\font\defaultthyphenchar}}
3200 % \fi}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly,

but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3201 \def\bbl@load@info#1{%
3202   \def\BabelBeforeIni##1##2{%
3203     \begingroup
3204       \bbl@read@ini{##1}0%
3205       \endinput           % babel- .tex may contain onlypreamble's
3206       \endgroup}%         boxed, to avoid extra spaces:
3207   {\bbl@input@texini{##1}}}
```

4.23. Numerals

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

```

3208 \def\bbl@setdigits#1#2#3#4#5{%
3209   \bbl@exp{%
3210     \def<\language name digits>####1{%       i.e., \langdigits
3211       \<bbl@digits@\language name>####1\\\@nil}%
3212       \let\<bbl@cntr@digits@\language name>\<\language name digits>%
3213       \def<\language name counter>####1{%     i.e., \langcounter
3214         \\\expandafter\<bbl@counter@\language name>%
3215         \\\csname c@####1\endcsname}%
3216       \def<\bbl@counter@\language name>####1{% i.e., \bbl@counter@lang
3217         \\\expandafter\<bbl@digits@\language name>%
3218         \\\number####1\\\@nil}}}%
3219 \def\bbl@tempa##1##2##3##4##5{%
3220   \bbl@exp{%    Wow, quite a lot of hashes! :- (
3221     \def<\bbl@digits@\language name>#####1{%
3222       \\\ifx#####1\\\@nil           % i.e., \bbl@digits@lang
3223       \\\else
3224         \\\ifx0#####1#1%
3225         \\\else\\\ifx1#####1#2%
3226         \\\else\\\ifx2#####1#3%
3227         \\\else\\\ifx3#####1#4%
3228         \\\else\\\ifx4#####1#5%
3229         \\\else\\\ifx5#####1#1%
3230         \\\else\\\ifx6#####1#2%
3231         \\\else\\\ifx7#####1#3%
3232         \\\else\\\ifx8#####1#4%
3233         \\\else\\\ifx9#####1#5%
3234         \\\else#####1%
3235         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3236         \\\expandafter\<bbl@digits@\language name>%
3237         \\\fi}}}%
3238   \bbl@tempa}
```

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

```

3239 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={ }
3240   \ifx\\#1%           % \ before, in case #1 is multiletter
3241     \bbl@exp{%
3242       \def\\\bbl@tempa####1{%
3243         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3244     \else
3245       \toks@\expandafter{\the\toks@\or #1}%
3246       \expandafter\bbl@buildifcase
3247     \fi}
```

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

```

3248 \newcommand\locaenumerat[2]{\bbl@cs{cnt@#1@\language}\{#2}}
3249 \def\bbl@locaecnt#1#2{\locaenumerat{#2}{#1}}
3250 \newcommand\localecounter[2]{%
3251   \expandafter\bbl@locaecnt
3252   \expandafter{\number\csname c@#2\endcsname}\{#1}}
3253 \def\bbl@alphnumerat#1#2{%
3254   \expandafter\bbl@alphnumerat@i\number#2 76543210\@@{#1}}
3255 \def\bbl@alphnumerat@i#1#2#3#4#5#6#7#8\@@#9{%
3256   \ifcase\car#8\@nil\or    % Currently <10000, but prepared for bigger
3257     \bbl@alphnumerat@ii{#9}000000#1\or
3258     \bbl@alphnumerat@ii{#9}000000#1#2\or
3259     \bbl@alphnumerat@ii{#9}000000#1#2#3\or
3260     \bbl@alphnumerat@ii{#9}000000#1#2#3#4\else
3261     \bbl@alphnum@invalid{>9999}%
3262   \fi}
3263 \def\bbl@alphnumerat@ii#1#2#3#4#5#6#7#8{%
3264   \bbl@ifunset{bbl@cnt@#1.F.\number#5#6#7#8@\language}%
3265     {\bbl@cs{cnt@#1.4@\language}\{#5}%
3266     \bbl@cs{cnt@#1.3@\language}\{#6}%
3267     \bbl@cs{cnt@#1.2@\language}\{#7}%
3268     \bbl@cs{cnt@#1.1@\language}\{#8}%
3269     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3270       \bbl@ifunset{bbl@cnt@#1.S.321@\language}\{}}%
3271     {\bbl@cs{cnt@#1.S.321@\language}\{}}%
3272   \fi}%
3273   {\bbl@cs{cnt@#1.F.\number#5#6#7#8@\language}\{}}
3274 \def\bbl@alphnum@invalid#1{%
3275   \bbl@error{alphabetic-too-large}{#1}\{}}

```

4.24. Casing

```

3276 \newcommand\BabelUppercaseMapping[3]{%
3277   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3278 \newcommand\BabelTitlecaseMapping[3]{%
3279   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3280 \newcommand\BabelLowercaseMapping[3]{%
3281   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

  The parser for casing and casing. (variant).
3282 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3283   \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3284 \else
3285   \def\bbl@utftocode#1{\expandafter\string#1}
3286 \fi
3287 \def\bbl@casemapping#1#2#3{% 1:variant
3288   \def\bbl@tempa##1 ##2{% Loop
3289     \bbl@casemapping@i{##1}%
3290     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3291   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3292   \def\bbl@tempe{0}% Mode (upper/lower...)
3293   \def\bbl@tempc{#3}% Casing list
3294   \expandafter\bbl@tempa\bbl@tempc\@empty}
3295 \def\bbl@casemapping@i#1{%
3296   \def\bbl@tempb{#1}%
3297   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3298     \@nameuse{regex_replace_all:nnN}%
3299     {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\{\0}\}\bbl@tempb
3300   \else
3301     \@nameuse{regex_replace_all:nnN}{.}{\{\0}\}\bbl@tempb % TODO. needed?
3302   \fi
3303   \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3304 \def\bbl@casemapping@ii#1#2#3\@@{%
3305   \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3306   \ifin@

```

```

3307 \edef\bbl@tempe{%
3308 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3309 \else
3310 \ifcase\bbl@tempe\relax
3311 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3312 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3313 \or
3314 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3315 \or
3316 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3317 \or
3318 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3319 \fi
3320 \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.

```

3321 \def\bbl@localeinfo#1#2{%
3322 \bbl@ifunset\bbl@info@#2}{#1}%
3323 {\bbl@ifunset\bbl@csname\bbl@info@#2\endcsname @\languagename}{#1}%
3324 {\bbl@cs{\csname\bbl@info@#2\endcsname @\languagename}}}%
3325 \newcommand\localeinfo[1]{%
3326 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3327 \bbl@afterelse\bbl@localeinfo{%
3328 \else
3329 \bbl@localeinfo
3330 {\bbl@error{no-ini-info}}{}}}%
3331 {#1}%
3332 \fi}
3333 % \@namedef\bbl@info@name.locale}{lname}
3334 \@namedef\bbl@info@tag.ini}{lini}
3335 \@namedef\bbl@info@name.english}{elname}
3336 \@namedef\bbl@info@name.opentype}{lname}
3337 \@namedef\bbl@info@tag.bcp47}{tbc}
3338 \@namedef\bbl@info@language.tag.bcp47}{lbc}
3339 \@namedef\bbl@info@tag.opentype}{lotf}
3340 \@namedef\bbl@info@script.name}{esname}
3341 \@namedef\bbl@info@script.name.opentype}{sname}
3342 \@namedef\bbl@info@script.tag.bcp47}{sbcp}
3343 \@namedef\bbl@info@script.tag.opentype}{sotf}
3344 \@namedef\bbl@info@region.tag.bcp47}{rbcp}
3345 \@namedef\bbl@info@variant.tag.bcp47}{vbcp}
3346 \@namedef\bbl@info@extension.t.tag.bcp47}{extt}
3347 \@namedef\bbl@info@extension.u.tag.bcp47}{extu}
3348 \@namedef\bbl@info@extension.x.tag.bcp47}{extx}

```

With version 3.75 `\BabelEnsureInfo` is executed always, but there is an option to disable it.

```

3349 <<More package options>> ≡
3350 \DeclareOption{ensureinfo=off}{}
3351 <</More package options>>
3352 \let\bbl@ensureinfo\@gobble
3353 \newcommand\BabelEnsureInfo{%
3354 \ifx\InputIfFileExists\@undefined\else
3355 \def\bbl@ensureinfo##1{%
3356 \bbl@ifunset\bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3357 \fi
3358 \bbl@foreach\bbl@loaded{%
3359 \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3360 \def\languagename{##1}%
3361 \bbl@ensureinfo{##1}}}%
3362 \@ifpackagewith{babel}{ensureinfo=off}{}%
3363 {\AtEndOfPackage{% Test for plain.

```

```
3364 \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}
```

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

```
3365 \newcommand\getlocaleproperty{%
3366 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3367 \def\bbl@getproperty@s#1#2#3{%
3368 \let#1\relax
3369 \def\bbl@elt##1##2##3{%
3370 \bbl@ifsamestring{##1/##2}{#3}%
3371 {\providecommand#1{##3}%
3372 \def\bbl@elt###1####2####3{}}}%
3373 {}}%
3374 \bbl@cs{inidata@#2}}%
3375 \def\bbl@getproperty@x#1#2#3{%
3376 \bbl@getproperty@s{#1}{#2}{#3}%
3377 \ifx#1\relax
3378 \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3379 \fi}
3380 \let\bbl@ini@loaded\@empty
3381 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3382 \def\ShowLocaleProperties#1{%
3383 \typeout{}}%
3384 \typeout{*** Properties for language '#1' ***}
3385 \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3386 \@nameuse{bbl@inidata@#1}%
3387 \typeout{*****}}
```

4.26. BCP 47 related commands

```
3388 \newif\ifbbl@bcpallowed
3389 \bbl@bcpallowedfalse
3390 \def\bbl@autoload@options{import}
3391 \def\bbl@provide@locale{%
3392 \ifx\babelprovide\@undefined
3393 \bbl@error{base-on-the-fly}{}{}%
3394 \fi
3395 \let\bbl@auxname\language\name % Still necessary. %^A TODO
3396 \bbl@ifunset{bbl@bcp@map@\language\name}{}% Move uplevel??
3397 {\edef\language\name{\@nameuse{bbl@bcp@map@\language\name}}}%
3398 \ifbbl@bcpallowed
3399 \expandafter\ifx\csname date\language\name\endcsname\relax
3400 \expandafter
3401 \bbl@bcplookup\language\name-\@empty-\@empty-\@empty@@
3402 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3403 \edef\language\name{\bbl@bcp@prefix\bbl@bcp}%
3404 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
3405 \expandafter\ifx\csname date\language\name\endcsname\relax
3406 \let\bbl@initoload\bbl@bcp
3407 \bbl@exp{\babelprovide[\bbl@autoload@options]{\language\name}}%
3408 \let\bbl@initoload\relax
3409 \fi
3410 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
3411 \fi
3412 \fi
3413 \fi
3414 \expandafter\ifx\csname date\language\name\endcsname\relax
3415 \IfFileExists{babel-\language.tex}%
3416 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language\name}}}%
3417 {}%
3418 \fi}
```

\LaTeX needs to know the BCP 47 codes for some features. For that, it expects `\BCPdata` to be defined.

While language, region, script, and variant are recognized, extension.⟨s⟩ for singletons may change.

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

```

3419 \providecommand\BCPdata{}
3420 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3421   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3422   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3423     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3424     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3425     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3426   \def\bbl@bcpdata@ii#1#2{%
3427     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3428     {\bbl@error{unknown-ini-field}{#1}{}}}%
3429     {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3430      {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3431 \fi
3432 \@namedef{bbl@info@casing.tag.bcp47}{casing}

```

5. Adjusting the Babel behavior

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

```

3433 \newcommand\babeladjust[1]{% TODO. Error handling.
3434   \bbl@forkv{#1}{%
3435     \bbl@ifunset{bbl@ADJ@##1@##2}%
3436     {\bbl@cs{ADJ@##1}{##2}}%
3437     {\bbl@cs{ADJ@##1@##2}}}
3438 %
3439 \def\bbl@adjust@lua#1#2{%
3440   \ifvmode
3441     \ifnum\currentgrouplevel=\z@
3442       \directlua{ Babel.#2 }%
3443       \expandafter\expandafter\expandafter\@gobble
3444     \fi
3445   \fi
3446   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3447 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3448   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3449 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3450   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3451 \@namedef{bbl@ADJ@bidi.text@on}{%
3452   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3453 \@namedef{bbl@ADJ@bidi.text@off}{%
3454   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3455 \@namedef{bbl@ADJ@bidi.math@on}{%
3456   \let\bbl@noamsmath\@empty}
3457 \@namedef{bbl@ADJ@bidi.math@off}{%
3458   \let\bbl@noamsmath\relax}
3459 %
3460 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3461   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3462 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3463   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3464 %
3465 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3466   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3467 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3468   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3469 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3470   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3471 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3472   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3473 \@namedef{bbl@ADJ@justify.arabic@on}{%

```

```

3474 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3475 \@namedef{bbl@ADJ@justify.arabic@off}{%
3476 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3477 %
3478 \def\bbl@adjust@layout#1{%
3479 \ifvmode
3480 #1%
3481 \expandafter\@gobble
3482 \fi
3483 {\bbl@error{layout-only-vertical}}{}}{}}}% Gobbled if everything went ok.
3484 \@namedef{bbl@ADJ@layout.tabular@on}{%
3485 \ifnum\bbl@tabular@mode=\tw@
3486 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3487 \else
3488 \chardef\bbl@tabular@mode\@ne
3489 \fi}
3490 \@namedef{bbl@ADJ@layout.tabular@off}{%
3491 \ifnum\bbl@tabular@mode=\tw@
3492 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3493 \else
3494 \chardef\bbl@tabular@mode\@z@
3495 \fi}
3496 \@namedef{bbl@ADJ@layout.lists@on}{%
3497 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3498 \@namedef{bbl@ADJ@layout.lists@off}{%
3499 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3500 %
3501 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3502 \bbl@bcpallowedtrue}
3503 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3504 \bbl@bcpallowedfalse}
3505 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3506 \def\bbl@bcp@prefix{#1}}
3507 \def\bbl@bcp@prefix{bcp47-}
3508 \@namedef{bbl@ADJ@autoload.options}#1{%
3509 \def\bbl@autoload@options{#1}}
3510 \def\bbl@autoload@bcptoptions{import}
3511 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3512 \def\bbl@autoload@bcptoptions{#1}}
3513 \newif\ifbbl@bcptname
3514 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3515 \bbl@bcptnametrue
3516 \BabelEnsureInfo}
3517 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3518 \bbl@bcptnamefalse}
3519 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3520 \directlua{ Babel.ignore_pre_char = function(node)
3521 return (node.lang == \the\csname \@nohyphenation\endcsname)
3522 end }}
3523 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3524 \directlua{ Babel.ignore_pre_char = function(node)
3525 return false
3526 end }}
3527 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3528 \def\bbl@ignoreinterchar{%
3529 \ifnum\language=\@nohyphenation
3530 \expandafter\@gobble
3531 \else
3532 \expandafter\@firstofone
3533 \fi}}
3534 \@namedef{bbl@ADJ@interchar.disable@off}{%
3535 \let\bbl@ignoreinterchar\@firstofone}
3536 \@namedef{bbl@ADJ@select.write@shift}{%

```

```

3537 \let\bbl@restorelastskip\relax
3538 \def\bbl@savelastskip{%
3539   \let\bbl@restorelastskip\relax
3540   \ifvmode
3541     \ifdim\lastskip=\z@
3542       \let\bbl@restorelastskip\nobreak
3543     \else
3544       \bbl@exp{%
3545         \def\\bbl@restorelastskip{%
3546           \skip@=\the\lastskip
3547           \\nobreak \vskip-\skip@ \vskip\skip@}}%
3548       \fi
3549   \fi}}
3550 \@namedef{bbl@ADJ@select.write@keep}{%
3551   \let\bbl@restorelastskip\relax
3552   \let\bbl@savelastskip\relax}
3553 \@namedef{bbl@ADJ@select.write@omit}{%
3554   \AddBabelHook{babel-select}{beforestart}{%
3555     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3556 \let\bbl@restorelastskip\relax
3557 \def\bbl@savelastskip##1\bbl@restorelastskip{}
3558 \@namedef{bbl@ADJ@select.encoding@off}{%
3559   \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.

```

3560 << *More package options >> ≡
3561 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3562 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3563 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3564 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3565 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3566 << /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.

```

3567 \bbl@trace{Cross referencing macros}
3568 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3569   \def\@newl@bel#1#2#3{%
3570     {\@safe@activetrue
3571       \bbl@ifunset{#1@#2}%
3572       \relax
3573       {\gdef\@multiplelabels{%
3574         \@latex@warning@no@line{There were multiply-defined labels}}%
3575         \@latex@warning@no@line{Label `#2' multiply defined}}%
3576       \global\@namedef{#1@#2}{#3}}}%

```

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

```

3577 \CheckCommand*\@testdef[3]{%
3578   \def\reserved@a{#3}%

```

```

3579 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3580 \else
3581 \@tempswattrue
3582 \fi}

```

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

```

3583 \def\@testdef#1#2#3{% TODO. With @samestring?
3584 \@safe@activestru
3585 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3586 \def\bbl@tempb{#3}%
3587 \@safe@activestru
3588 \ifx\bbl@tempa\relax
3589 \else
3590 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3591 \fi
3592 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3593 \ifx\bbl@tempa\bbl@tempb
3594 \else
3595 \@tempswattrue
3596 \fi}
3597 \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.

```

3598 \bbl@xin@{R}\bbl@opt@safe
3599 \ifin@
3600 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3601 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3602 {\expandafter\strip@prefix\meaning\ref}%
3603 \ifin@
3604 \bbl@redefine\@kernel@ref#1{%
3605 \@safe@activestru\org@@kernel@ref{#1}\@safe@activestru}
3606 \bbl@redefine\@kernel@pageref#1{%
3607 \@safe@activestru\org@@kernel@pageref{#1}\@safe@activestru}
3608 \bbl@redefine\@kernel@sref#1{%
3609 \@safe@activestru\org@@kernel@sref{#1}\@safe@activestru}
3610 \bbl@redefine\@kernel@spageref#1{%
3611 \@safe@activestru\org@@kernel@spageref{#1}\@safe@activestru}
3612 \else
3613 \bbl@redefinero\ref#1{%
3614 \@safe@activestru\org@ref{#1}\@safe@activestru}
3615 \bbl@redefinero\pageref#1{%
3616 \@safe@activestru\org@pageref{#1}\@safe@activestru}
3617 \fi
3618 \else
3619 \let\org@ref\ref
3620 \let\org@pageref\pageref
3621 \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.

```

3622 \bbl@xin@{B}\bbl@opt@safe
3623 \ifin@
3624 \bbl@redefine\@citex[#1]#2{%

```



```

3625 \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetruefalse
3626 \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.)

```

3627 \AtBeginDocument{%
3628 \ifpackageloaded{natbib}{%
3629 \def\@citex[#1][#2]#3{%
3630 \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activetruefalse
3631 \org@citex[#1][#2]{\bbl@tempa}}%
3632 }{}}

```

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

```

3633 \AtBeginDocument{%
3634 \ifpackageloaded{cite}{%
3635 \def\@citex[#1]#2{%
3636 \@safe@activetrue\org@citex[#1][#2]\@safe@activetruefalse}%
3637 }{}}

```

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

```

3638 \bbl@redefine\nocite#1{%
3639 \@safe@activetrue\org@citex[#1]\@safe@activetruefalse}

```

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

```

3640 \bbl@redefine\bibcite{%
3641 \bbl@cite@choice
3642 \bibcite}

```

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

```

3643 \def\bbl@bibcite#1#2{%
3644 \org@bibcite{#1}{\@safe@activetruefalse#2}}

```

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

```

3645 \def\bbl@cite@choice{%
3646 \global\let\bibcite\bbl@bibcite
3647 \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3648 \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3649 \global\let\bbl@cite@choice\relax}

```

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

```

3650 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3651 \bbl@redefine\@bibitem#1{%
3652   \safe@activetrue\org@bibitem{#1}\safe@activesfalse}
3653 \else
3654   \let\org@nocite\nocite
3655   \let\org@@citex@citex
3656   \let\org@bibcite\bibcite
3657   \let\org@bibitem\@bibitem
3658 \fi

```

5.2. Layout

```

3659 \newcommand\BabelPatchSection[1]{%
3660   \@ifundefined{#1}{}{%
3661     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3662     \@namedef{#1}{%
3663       \@ifstar{\bbl@presec@#1}%
3664       {\@dblarg{\bbl@presec@#1}}}%
3665 \def\bbl@presec@#1[#2]#3{%
3666   \bbl@exp{%
3667     \\\select@language@x{\bbl@main@language}%
3668     \\\bbl@cs{sspre@#1}%
3669     \\\bbl@cs{ss@#1}%
3670     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3671     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3672     \\\select@language@x{\language}}%
3673 \def\bbl@presec@#1#2{%
3674   \bbl@exp{%
3675     \\\select@language@x{\bbl@main@language}%
3676     \\\bbl@cs{sspre@#1}%
3677     \\\bbl@cs{ss@#1}%
3678     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3679     \\\select@language@x{\language}}%
3680 \IfBabelLayout{sectioning}%
3681   {\BabelPatchSection{part}%
3682    \BabelPatchSection{chapter}%
3683    \BabelPatchSection{section}%
3684    \BabelPatchSection{subsection}%
3685    \BabelPatchSection{subsubsection}%
3686    \BabelPatchSection{paragraph}%
3687    \BabelPatchSection{subparagraph}%
3688   \def\babel@toc#1{%
3689     \select@language@x{\bbl@main@language}}}%
3690 \IfBabelLayout{captions}%
3691   {\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.

```

3692 \bbl@trace{Marks}
3693 \IfBabelLayout{sectioning}
3694   {\ifx\bbl@opt@headfoot\@nnil
3695     \g@addto@macro\@resetactivechars{%
3696       \set@typeset@protect
3697       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3698       \let\protect\noexpand
3699       \ifcase\bbl@bidimode\else % Only with bidi. See also above

```

```

3700         \edef\thepage{%
3701             \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3702         \fi}%
3703     \fi}
3704     {\ifbbl@single\else
3705         \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3706             \markright#1{%
3707                 \bbl@ifblank{#1}%
3708                 {\org@markright{}}}%
3709                 {\toks@{#1}%
3710                 \bbl@exp{%
3711                     \\org@markright{\\protect\\foreignlanguage{\language}%
3712                     {\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.)

```

3713     \ifx\@mkboth\markboth
3714         \def\bbl@tempc{\let\@mkboth\markboth}%
3715     \else
3716         \def\bbl@tempc{%
3717             \fi
3718             \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3719                 \markboth#1#2{%
3720                     \protected@edef\bbl@tempb##1{%
3721                         \protect\foreignlanguage
3722                         {\language}{\protect\bbl@restore@actives##1}}%
3723                     \bbl@ifblank{#1}%
3724                     {\toks@{}}%
3725                     {\toks@\expandafter{\bbl@tempb{#1}}}%
3726                     \bbl@ifblank{#2}%
3727                     {\@temptokena{}}%
3728                     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3729                     \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3730                     \bbl@tempc
3731             \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.

```

3732 \bbl@trace{Preventing clashes with other packages}

```

```

3733 \ifx\org@ref\@undefined\else
3734 \bbl@xin@{R}\bbl@opt@safe
3735 \ifin@
3736 \AtBeginDocument{%
3737 \ifpackageloaded{ifthen}{%
3738 \bbl@redefine@long\ifthenelse#1#2#3{%
3739 \let\bbl@temp@pref\pageref
3740 \let\pageref\org@pageref
3741 \let\bbl@temp@ref\ref
3742 \let\ref\org@ref
3743 \@safe@activestrue
3744 \org@ifthenelse{#1}%
3745 {\let\pageref\bbl@temp@pref
3746 \let\ref\bbl@temp@ref
3747 \@safe@activesfalse
3748 #2}%
3749 {\let\pageref\bbl@temp@pref
3750 \let\ref\bbl@temp@ref
3751 \@safe@activesfalse
3752 #3}%
3753 }%
3754 }{}%
3755 }
3756 \fi

```

5.4.2. varioref

\@@vpageref

\vrefpagenum

\Ref When the package varioref is in use we need to modify its internal command \@@vpageref in order to prevent problems when an active character ends up in the argument of \vref. The same needs to happen for \vrefpagenum.

```

3757 \AtBeginDocument{%
3758 \ifpackageloaded{varioref}{%
3759 \bbl@redefine\@@vpageref#1[#2]#3{%
3760 \@safe@activestrue
3761 \org@@vpageref{#1}[#2]#3}%
3762 \@safe@activesfalse}%
3763 \bbl@redefine\vrefpagenum#1#2{%
3764 \@safe@activestrue
3765 \org@vrefpagenum{#1}#2}%
3766 \@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.

```

3767 \expandafter\def\csname Ref \endcsname#1{%
3768 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3769 }{}%
3770 }
3771 \fi

```

5.4.3. hhl ine

\hhl ine Delaying the activation of the shorthand characters has introduced a problem with the hhl ine 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.

```

3772 \AtEndOfPackage{%

```

```

3773 \AtBeginDocument{%
3774   \@ifpackageloaded{hhline}%
3775     {\expandafter\ifx\cename normal@char\string:\endcsname\relax
3776     \else
3777       \makeatletter
3778       \def\@currname{hhline}\input{hhline.sty}\makeatother
3779       \fi}%
3780   {}}

```

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

```

3781 \def\substitutefontfamily#1#2#3{%
3782   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3783   \immediate\write15{%
3784     \string\ProvidesFile{#1#2.fd}%
3785     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3786     \space generated font description file]^J
3787     \string\DeclareFontFamily{#1}{#2}{}}^J
3788     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3789     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3790     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3791     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3792     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3793     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3794     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3795     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3796   }%
3797   \closeout15
3798 }
3799 \@onlypreamble\substitutefontfamily

```

5.5. Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings are currently stored in `\@fontenc@load@list`. If a non-ASCII has been loaded, we define versions of `\TeX` and `\LaTeX` for them using `\ensureascii`. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensureascii

```

3800 \bbl@trace{Encoding and fonts}
3801 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3802 \newcommand\BabelNonText{TS1,T3,TS3}
3803 \let\org@TeX\TeX
3804 \let\org@LaTeX\LaTeX
3805 \let\ensureascii@firstofone
3806 \let\asciiencoding\@empty
3807 \AtBeginDocument{%
3808   \def\@elt#1{,#1,}%
3809   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3810   \let\@elt\relax
3811   \let\bbl@tempb\@empty
3812   \def\bbl@tempc{OT1}%
3813   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3814     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3815   \bbl@foreach\bbl@tempa{%
3816     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3817     \ifin@
3818       \def\bbl@tempb{#1}% Store last non-ascii
3819     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3820     \ifin@else

```

```

3821      \def\bbl@tempc{#1}% Store last ascii
3822      \fi
3823      \fi}%
3824      \ifx\bbl@tempb\@empty\else
3825      \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3826      \ifin@else
3827      \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3828      \fi
3829      \let\asciientcoding\bbl@tempc
3830      \renewcommand\ensureascii[1]{%
3831      {\fontencoding{\asciientcoding}\selectfont#1}}%
3832      \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3833      \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3834      \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.

```

3835 \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.

```

3836 \AtBeginDocument{%
3837   \@ifpackageloaded{fontspec}%
3838   {\xdef\latinencoding{%
3839     \ifx\UTFencname\@undefined
3840     EU\ifcase\bbl@engine\or2\or1\fi
3841     \else
3842     \UTFencname
3843     \fi}}%
3844   {\gdef\latinencoding{OT1}%
3845     \ifx\cf@encoding\bbl@t@one
3846     \xdef\latinencoding{\bbl@t@one}%
3847     \else
3848     \def\@elt#1{,#1,}%
3849     \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3850     \let\@elt\relax
3851     \bbl@xin@{,T1,}\bbl@tempa
3852     \ifin@
3853     \xdef\latinencoding{\bbl@t@one}%
3854     \fi
3855     \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.

```

3856 \DeclareRobustCommand{\latintext}{%
3857   \fontencoding{\latinencoding}\selectfont
3858   \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.

```

3859 \ifx\@undefined\DeclareTextFontCommand
3860   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3861 \else
3862   \DeclareTextFontCommand{\textlatin}{\latintext}
3863 \fi

```

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

```
3864 \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.

```
3865 \bbl@trace{Loading basic (internal) bidi support}
3866 \ifodd\bbl@engine
3867 \else % TODO. Move to txtbabel. Any xe+lua bidi
3868   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3869     \bbl@error{bidi-only-lua}{\}\}\}\}%
3870     \let\bbl@beforeforeign\leavevmode
3871     \AtEndOfPackage{%
3872       \EnableBabelHook{babel-bidi}%
3873       \bbl@xebidipar}
3874 \fi\fi
3875 \def\bbl@loadxebidi#1{%
3876   \ifx\RTLfootnotetext\@undefined
3877     \AtEndOfPackage{%
3878       \EnableBabelHook{babel-bidi}%
3879       \ifx\fontspec\@undefined
3880         \usepackage{fontspec}% bidi needs fontspec
3881       \fi
3882       \usepackage#1{bidi}%
3883       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3884       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3885         \ifnum\@nameuse{bbl@wdir@languagename}=\tw@ % 'AL' bidi
3886           \bbl@digitsdotdash % So ignore in 'R' bidi
3887         \fi}}%
3888   \fi}
3889 \ifnum\bbl@bidimode>200 % Any xe bidi=
3890   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3891     \bbl@tentative{bidi=bidi}
3892     \bbl@loadxebidi{}
3893   \or
3894     \bbl@loadxebidi{[rldocument]}
3895   \or
3896     \bbl@loadxebidi{}
3897   \fi
3898 \fi
3899 \fi
3900 % TODO? Separate:
```

```

3901 \ifnum\bbbl@bidimode=\@ne % bidi=default
3902 \let\bbbl@beforeforeign\leavevmode
3903 \ifodd\bbbl@engine % lua
3904 \newattribute\bbbl@attr@dir
3905 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
3906 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
3907 \fi
3908 \AtEndOfPackage{%
3909 \EnableBabelHook{babel-bidi}% pdf/lua/xe
3910 \ifodd\bbbl@engine\else % pdf/xe
3911 \bbbl@xebidipar
3912 \fi}
3913 \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.

```

3914 \bbbl@trace{Macros to switch the text direction}
3915 \def\bbbl@alscripts{,Arabic,Syriac,Thaana,}
3916 \def\bbbl@rscripts{%
3917 ,Garay,Todhri,Imperial Aramaic,Avestan,Cypriot,Elymaic,Hatran,Hebrew,%
3918 Old Hungarian,Kharoshthi,Lydian,Mandaean,Manichaeen,Mende Kikakui,%
3919 Meroitic Cursive,Meroitic,Old North Arabian,Nabataean,N'Ko,%
3920 Old Turkic,Orkhon,Palmyrene,Inscriptional Pahlavi,Psalter Pahlavi,%
3921 Phoenician,Inscriptional Parthian,Hanifi,Samaritan,Old Sogdian,%
3922 Old South Arabian,Yezidi,}%
3923 \def\bbbl@provide@dirs#1{%
3924 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
3925 \ifin@
3926 \global\bbbl@csarg\chardef{wdir@#1}\@ne
3927 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
3928 \ifin@
3929 \global\bbbl@csarg\chardef{wdir@#1}\tw@
3930 \fi
3931 \else
3932 \global\bbbl@csarg\chardef{wdir@#1}\z@
3933 \fi
3934 \ifodd\bbbl@engine
3935 \bbbl@csarg\ifcase{wdir@#1}%
3936 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3937 \or
3938 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3939 \or
3940 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3941 \fi
3942 \fi}
3943 \def\bbbl@switchdir{%
3944 \bbbl@ifunset{\bbbl@sys@\language name}{\bbbl@provide@sys@\language name}}}%
3945 \bbbl@ifunset{\bbbl@wdir@\language name}{\bbbl@provide@dirs@\language name}}}%
3946 \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}}%
3947 \def\bbbl@setdirs#1{% TODO - math
3948 \ifcase\bbbl@select@type % TODO - strictly, not the right test
3949 \bbbl@bodydir{#1}%
3950 \bbbl@pardir{#1}% <- Must precede \bbbl@textdir
3951 \fi
3952 \bbbl@textdir{#1}}
3953 \ifnum\bbbl@bidimode>\z@
3954 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
3955 \DisableBabelHook{babel-bidi}
3956 \fi

```

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

```

3957 \ifodd\bbbl@engine % luatex=1
3958 \else % pdftex=0, xetex=2

```



```

3959 \newcount\bbl@dirlevel
3960 \chardef\bbl@thetextdir\z@
3961 \chardef\bbl@thepardir\z@
3962 \def\bbl@textdir#1{%
3963   \ifcase#1\relax
3964     \chardef\bbl@thetextdir\z@
3965     \@nameuse{setlatin}%
3966     \bbl@textdir@i\beginL\endL
3967   \else
3968     \chardef\bbl@thetextdir@ne
3969     \@nameuse{setnonlatin}%
3970     \bbl@textdir@i\beginR\endR
3971   \fi}
3972 \def\bbl@textdir@i#1#2{%
3973   \ifhmode
3974     \ifnum\currentgrouplevel>\z@
3975       \ifnum\currentgrouplevel=\bbl@dirlevel
3976         \bbl@error{multiple-bidi}{\}\}%
3977         \bgroup\aftergroup#2\aftergroup\egroup
3978       \else
3979         \ifcase\currentgrouptype\or % 0 bottom
3980           \aftergroup#2% 1 simple {}
3981         \or
3982           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3983         \or
3984           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
3985         \or\or\or % vbox vtop align
3986         \or
3987           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3988         \or\or\or\or\or % output math disc insert vcent mathchoice
3989         \or
3990           \aftergroup#2% 14 \begingroup
3991         \else
3992           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3993         \fi
3994       \fi
3995       \bbl@dirlevel\currentgrouplevel
3996     \fi
3997     #1%
3998   \fi}
3999 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4000 \let\bbl@bodydir@gobble
4001 \let\bbl@pagedir@gobble
4002 \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).

```

4003 \def\bbl@xebidipar{%
4004   \let\bbl@xebidipar\relax
4005   \TeXeTstate@ne
4006   \def\bbl@xeeverypar{%
4007     \ifcase\bbl@thepardir
4008       \ifcase\bbl@thetextdir\else\beginR\fi
4009     \else
4010       {\setbox\z@\lastbox\beginR\box\z@}%
4011     \fi}%
4012   \AddToHook{para/begin}{\bbl@xeeverypar}}
4013 \ifnum\bbl@bidimode>200 % Any xe bidi=
4014   \let\bbl@textdir@i@gobbletwo
4015   \let\bbl@xebidipar@empty
4016   \AddBabelHook{bidi}{foreign}{%
4017     \ifcase\bbl@thetextdir

```

```

4018     \BabelWrapText{\LR{##1}}%
4019     \else
4020     \BabelWrapText{\RL{##1}}%
4021     \fi}
4022     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4023   \fi
4024 \fi

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

4025 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4026 \AtBeginDocument{%
4027   \ifx\pdfstringdefDisableCommands@undefined\else
4028   \ifx\pdfstringdefDisableCommands\relax\else
4029   \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4030   \fi
4031 \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.

```

4032 \bbl@trace{Local Language Configuration}
4033 \ifx\loadlocalcfg@undefined
4034   \ifpackagewith{babel}{noconfigs}%
4035   {\let\loadlocalcfg@gobble}%
4036   {\def\loadlocalcfg#1{%
4037     \InputIfFileExists{#1.cfg}%
4038     {\typeout{*****^}%
4039              * Local config file #1.cfg used^^}%
4040     *}}%
4041   \@empty}}
4042 \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).

```

4043 \bbl@trace{Language options}
4044 \let\bbl@afterlang\relax
4045 \let\babelmodifiers\relax
4046 \let\bbl@loaded\@empty
4047 \def\bbl@load@language#1{%
4048   \InputIfFileExists{#1.ldf}%
4049   {\edef\bbl@loaded{\CurrentOption
4050     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4051     \expandafter\let\expandafter\bbl@afterlang
4052     \csname\CurrentOption.ldf-h@k\endcsname
4053     \expandafter\let\expandafter\babelmodifiers
4054     \csname bbl@mod@\CurrentOption\endcsname
4055     \bbl@exp{\AtBeginDocument{%
4056       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4057   {\IfFileExists{babel-#1.tex}%
4058     {\def\bbl@tempa{%
4059       .\\There is a locale ini file for this language.\\%
4060       If it's the main language, try adding `provide=*'\%
4061       to the babel package options}}%
4062     {\let\bbl@tempa\empty}%
4063     \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.

```

4064 \def\bbl@try@load@lang#1#2#3{%
4065   \IfFileExists{\CurrentOption.ldf}%
4066     {\bbl@load@language{\CurrentOption}}%
4067     {#1\bbl@load@language{#2}#3}}
4068 %
4069 \DeclareOption{friulian}{\bbl@try@load@lang{}{friulan}{}}
4070 \DeclareOption{hebrew}{%
4071   \ifcase\bbl@engine\or
4072     \bbl@error{only-pdftex-lang}{hebrew}{luatex}{}%
4073   \fi
4074   \input{rlbabel.def}%
4075   \bbl@load@language{hebrew}}
4076 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4077 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4078 % \DeclareOption{nothernkurdish}{\bbl@try@load@lang{}{kurmanji}{}}
4079 \DeclareOption{polutonikogreek}{%
4080   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4081 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4082 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4083 \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.

```

4084 \ifx\bbl@opt@config\@nnil
4085   \ifpackagewith{babel}{noconfigs}{}%
4086     {\InputIfFileExists{bblopts.cfg}%
4087       {\typeout{*****^J%
4088         * Local config file bblopts.cfg used^^J%
4089         *}}%
4090       {}}%
4091 \else
4092   \InputIfFileExists{\bbl@opt@config.cfg}%
4093     {\typeout{*****^J%
4094       * Local config file \bbl@opt@config.cfg used^^J%
4095       *}}%
4096     {\bbl@error{config-not-found}{}}}%
4097 \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).

```

4098 \def\bbl@tempf{,}
4099 \bbl@foreach\@raw@classoptionslist{%
4100   \in@{=}{#1}%
4101   \ifin@else
4102     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty}%
4103   \fi}
4104 \ifx\bbl@opt@main\@nnil
4105   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4106     \let\bbl@tempb\@empty
4107     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4108     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%

```

```

4109 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4110 \ifx\bbl@opt@main\@nnil % i.e., if not yet assigned
4111 \ifodd\bbl@iniflag % = *=
4112 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4113 \else % n +=
4114 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4115 \fi
4116 \fi}%
4117 \fi
4118 \else
4119 \bbl@info{Main language set with 'main='. Except if you have\\%
4120 problems, prefer the default mechanism for setting\\%
4121 the main language, i.e., as the last declared.\\%
4122 Reported}
4123 \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`).

```

4124 \ifx\bbl@opt@main\@nnil\else
4125 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4126 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4127 \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.

```

4128 \bbl@foreach\bbl@language@opts{%
4129 \def\bbl@tempa{#1}%
4130 \ifx\bbl@tempa\bbl@opt@main\else
4131 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4132 \bbl@ifunset{ds@#1}%
4133 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4134 {}%
4135 \else % + * (other = ini)
4136 \DeclareOption{#1}{%
4137 \bbl@ldfinit
4138 \babelprovide[@import]{#1}% %%%
4139 \bbl@afterldf{}}%
4140 \fi
4141 \fi}
4142 \bbl@foreach\bbl@tempf{%
4143 \def\bbl@tempa{#1}%
4144 \ifx\bbl@tempa\bbl@opt@main\else
4145 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4146 \bbl@ifunset{ds@#1}%
4147 {\IfFileExists{#1.ldf}%
4148 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4149 {}}%
4150 {}%
4151 \else % + * (other = ini)
4152 \IfFileExists{babel-#1.tex}%
4153 {\DeclareOption{#1}{%
4154 \bbl@ldfinit
4155 \babelprovide[@import]{#1}% %%%
4156 \bbl@afterldf{}}}%
4157 {}%
4158 \fi
4159 \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 `\TeX` 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):

```

4160 \NewHook{babel/presets}

```

```

4161 \UseHook{babel/presets}
4162 \def\AfterBabelLanguage#1{%
4163   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4164 \DeclareOption*{}
4165 \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.

```

4166 \bbl@trace{Option 'main'}
4167 \ifx\bbl@opt@main\@nnil
4168   \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4169   \let\bbl@tempc\@empty
4170   \edef\bbl@templ{\,\bbl@loaded,}
4171   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4172   \bbl@for\bbl@tempb\bbl@tempa{%
4173     \edef\bbl@tempd{\,\bbl@tempb,}%
4174     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4175     \bbl@xin{\bbl@tempd}{\bbl@templ}%
4176     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4177   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4178   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4179   \ifx\bbl@tempb\bbl@tempc\else
4180     \bbl@warning{%
4181       Last declared language option is '\bbl@tempc',\%
4182       but the last processed one was '\bbl@tempb'.\%
4183       The main language can't be set as both a global\%
4184       and a package option. Use 'main=\bbl@tempc' as\%
4185       option. Reported}
4186   \fi
4187 \else
4188   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4189     \bbl@ldfinit
4190     \let\CurrentOption\bbl@opt@main
4191     \bbl@exp{% \bbl@opt@provide = empty if *
4192       \\ \babelprovide
4193         [\bbl@opt@provide,@import,main]% %%%
4194         {\bbl@opt@main}}%
4195     \bbl@afterldf{}
4196     \DeclareOption{\bbl@opt@main}{}
4197   \else % case 0,2 (main is ldf)
4198     \ifx\bbl@loadmain\relax
4199       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4200     \else
4201       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4202     \fi
4203     \ExecuteOptions{\bbl@opt@main}
4204     \@namedef{ds@\bbl@opt@main}{}%
4205   \fi
4206   \DeclareOption*{}
4207   \ProcessOptions*
4208 \fi
4209 \bbl@exp{%
4210   \\ \AtBeginDocument{\\ \bbl@usehooks@lang/{ }\{begindocument\}{ }\{ }\}%
4211   \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.

```

4212 \ifx\bbl@main@language\@undefined
4213   \bbl@info{%
4214     You haven't specified a language as a class or package\%

```

```

4215 option. I'll load 'nil'. Reported}
4216 \bbl@load@language{nil}
4217 \fi
4218 </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`

```

4219 <{*kernel}
4220 \let\bbl@onlyswitch\@empty
4221 \input babel.def
4222 \let\bbl@onlyswitch\@undefined
4223 </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.

```

4224 <{*errors}
4225 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4226 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4227 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4228 \catcode`\@=11 \catcode`\^=7
4229 %
4230 \ifx\MessageBreak\@undefined
4231 \gdef\bbl@error@i#1#2{%
4232 \begingroup
4233 \newlinechar=`^^J
4234 \def\{^J(babel) }%
4235 \errhelp{#2}\errmessage{\{#1}%
4236 \endgroup}
4237 \else
4238 \gdef\bbl@error@i#1#2{%
4239 \begingroup
4240 \def\{\MessageBreak}%
4241 \PackageError{babel}{#1}{#2}%
4242 \endgroup}
4243 \fi
4244 \def\bbl@errmessage#1#2#3{%
4245 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4246 \bbl@error@i{#2}{#3}}
4247 % Implicit #2#3#4:
4248 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4249 %
4250 \bbl@errmessage{not-yet-available}
4251 {Not yet available}%
4252 {Find an armchair, sit down and wait}
4253 \bbl@errmessage{bad-package-option}%
4254 {Bad option '#1=#2'. Either you have misspelled the\%

```

```

4255     key or there is a previous setting of '#1'. Valid\\%
4256     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4257     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4258     {See the manual for further details.}
4259 \bbl@errmessage{base-on-the-fly}
4260     {For a language to be defined on the fly 'base'\\%
4261     is not enough, and the whole package must be\\%
4262     loaded. Either delete the 'base' option or\\%
4263     request the languages explicitly}%
4264     {See the manual for further details.}
4265 \bbl@errmessage{undefined-language}
4266     {You haven't defined the language '#1' yet.\\%
4267     Perhaps you misspelled it or your installation\\%
4268     is not complete}%
4269     {Your command will be ignored, type <return> to proceed}
4270 \bbl@errmessage{shorthand-is-off}
4271     {I can't declare a shorthand turned off (\string#2)}
4272     {Sorry, but you can't use shorthands which have been\\%
4273     turned off in the package options}
4274 \bbl@errmessage{not-a-shorthand}
4275     {The character '\string #1' should be made a shorthand character;\\%
4276     add the command \string\usesshorthands\string{#1\string} to
4277     the preamble.\\%
4278     I will ignore your instruction}%
4279     {You may proceed, but expect unexpected results}
4280 \bbl@errmessage{not-a-shorthand-b}
4281     {I can't switch '\string#2' on or off--not a shorthand}%
4282     {This character is not a shorthand. Maybe you made\\%
4283     a typing mistake? I will ignore your instruction.}
4284 \bbl@errmessage{unknown-attribute}
4285     {The attribute #2 is unknown for language #1.}%
4286     {Your command will be ignored, type <return> to proceed}
4287 \bbl@errmessage{missing-group}
4288     {Missing group for string \string#1}%
4289     {You must assign strings to some category, typically\\%
4290     captions or extras, but you set none}
4291 \bbl@errmessage{only-lua-xe}
4292     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4293     {Consider switching to these engines.}
4294 \bbl@errmessage{only-lua}
4295     {This macro is available only in LuaLaTeX}%
4296     {Consider switching to that engine.}
4297 \bbl@errmessage{unknown-provide-key}
4298     {Unknown key '#1' in \string\babelprovide}%
4299     {See the manual for valid keys}%
4300 \bbl@errmessage{unknown-mapfont}
4301     {Option '\bbl@KVP@mapfont' unknown for\\%
4302     mapfont. Use 'direction'}%
4303     {See the manual for details.}
4304 \bbl@errmessage{no-ini-file}
4305     {There is no ini file for the requested language\\%
4306     (#1: \language). Perhaps you misspelled it or your\\%
4307     installation is not complete}%
4308     {Fix the name or reinstall babel.}
4309 \bbl@errmessage{digits-is-reserved}
4310     {The counter name 'digits' is reserved for mapping\\%
4311     decimal digits}%
4312     {Use another name.}
4313 \bbl@errmessage{limit-two-digits}
4314     {Currently two-digit years are restricted to the\\%
4315     range 0-9999}%
4316     {There is little you can do. Sorry.}
4317 \bbl@errmessage{alphabetic-too-large}

```

```

4318 {Alphabetic numeral too large (#1)}%
4319 {Currently this is the limit.}
4320 \bbl@errmessage{no-ini-info}
4321 {I've found no info for the current locale.\\%
4322   The corresponding ini file has not been loaded\\%
4323   Perhaps it doesn't exist}%
4324 {See the manual for details.}
4325 \bbl@errmessage{unknown-ini-field}
4326 {Unknown field '#1' in \string\BCPdata.\\%
4327   Perhaps you misspelled it}%
4328 {See the manual for details.}
4329 \bbl@errmessage{unknown-locale-key}
4330 {Unknown key for locale '#2':\\%
4331   #3\\%
4332   \string#1 will be set to \string\relax}%
4333 {Perhaps you misspelled it.}%
4334 \bbl@errmessage{adjust-only-vertical}
4335 {Currently, #1 related features can be adjusted only\\%
4336   in the main vertical list}%
4337 {Maybe things change in the future, but this is what it is.}
4338 \bbl@errmessage{layout-only-vertical}
4339 {Currently, layout related features can be adjusted only\\%
4340   in vertical mode}%
4341 {Maybe things change in the future, but this is what it is.}
4342 \bbl@errmessage{bidi-only-lua}
4343 {The bidi method 'basic' is available only in\\%
4344   luatex. I'll continue with 'bidi=default', so\\%
4345   expect wrong results}%
4346 {See the manual for further details.}
4347 \bbl@errmessage{multiple-bidi}
4348 {Multiple bidi settings inside a group}%
4349 {I'll insert a new group, but expect wrong results.}
4350 \bbl@errmessage{unknown-package-option}
4351 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4352   or the language definition file \CurrentOption.ldf\\%
4353   was not found%
4354   \bbl@tempa}
4355 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4356   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4357   headfoot=, strings=, config=, hyphenmap=, or a language name.}
4358 \bbl@errmessage{config-not-found}
4359 {Local config file '\bbl@opt@config.cfg' not found}%
4360 {Perhaps you misspelled it.}
4361 \bbl@errmessage{late-after-babel}
4362 {Too late for \string\AfterBabelLanguage}%
4363 {Languages have been loaded, so I can do nothing}
4364 \bbl@errmessage{double-hyphens-class}
4365 {Double hyphens aren't allowed in \string\babelcharclass\\%
4366   because it's potentially ambiguous}%
4367 {See the manual for further info}
4368 \bbl@errmessage{unknown-interchar}
4369 {'#1' for '\language' cannot be enabled.\\%
4370   Maybe there is a typo}%
4371 {See the manual for further details.}
4372 \bbl@errmessage{unknown-interchar-b}
4373 {'#1' for '\language' cannot be disabled.\\%
4374   Maybe there is a typo}%
4375 {See the manual for further details.}
4376 \bbl@errmessage{charproperty-only-vertical}
4377 {\string\babelcharproperty\space can be used only in\\%
4378   vertical mode (preamble or between paragraphs)}%
4379 {See the manual for further info}
4380 \bbl@errmessage{unknown-char-property}

```



```

4381 {No property named '#2'. Allowed values are\\%
4382 direction (bc), mirror (bmg), and linebreak (lb)}%
4383 {See the manual for further info}
4384 \bbl@errmessage{bad-transform-option}
4385 {Bad option '#1' in a transform.\\%
4386 I'll ignore it but expect more errors}%
4387 {See the manual for further info.}
4388 \bbl@errmessage{font-conflict-transforms}
4389 {Transforms cannot be re-assigned to different\\%
4390 fonts. The conflict is in '\bbl@kv@label'.\\%
4391 Apply the same fonts or use a different label}%
4392 {See the manual for further details.}
4393 \bbl@errmessage{transform-not-available}
4394 {'#1' for '\language' cannot be enabled.\\%
4395 Maybe there is a typo or it's a font-dependent transform}%
4396 {See the manual for further details.}
4397 \bbl@errmessage{transform-not-available-b}
4398 {'#1' for '\language' cannot be disabled.\\%
4399 Maybe there is a typo or it's a font-dependent transform}%
4400 {See the manual for further details.}
4401 \bbl@errmessage{year-out-range}
4402 {Year out of range.\\%
4403 The allowed range is #1}%
4404 {See the manual for further details.}
4405 \bbl@errmessage{only-pdftex-lang}
4406 {The '#1' ldf style doesn't work with #2,\\%
4407 but you can use the ini locale instead.\\%
4408 Try adding 'provide=' to the option list. You may\\%
4409 also want to set 'bidi=' to some value}%
4410 {See the manual for further details.}
4411 \bbl@errmessage{hyphenmins-args}
4412 {\string\babelhyphenmins\ accepts either the optional\\%
4413 argument or the star, but not both at the same time}%
4414 {See the manual for further details.}
4415 </errors>
4416 <*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.

```

4417 <@Make sure ProvidesFile is defined@>
4418 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4419 \xdef\bbl@format{\jobname}
4420 \def\bbl@version{<@version@>}
4421 \def\bbl@date{<@date@>}
4422 \ifx\AtBeginDocument\undefined
4423 \def\@empty{}
4424 \fi
4425 <@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.

```

4426 \def\process@line#1#2 #3 #4 {%
4427 \ifx=#1%
4428 \process@synonym{#2}%
4429 \else
4430 \process@language{#1#2}{#3}{#4}%
4431 \fi

```

```
4432 \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.

```
4433 \toks@{}
4434 \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.

```
4435 \def\process@synonym#1{%
4436   \ifnum\last@language=\m@ne
4437     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4438   \else
4439     \expandafter\chardef\csname l@#1\endcsname\last@language
4440     \wlog{\string\l@#1=\string\language\the\last@language}%
4441     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4442       \csname\language\hyphenmins\endcsname
4443     \let\bbl@elt\relax
4444     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4445   \fi}
```

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

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

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

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

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

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

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

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

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

```
4446 \def\process@language#1#2#3{%
4447   \expandafter\addlanguage\csname l@#1\endcsname
4448   \expandafter\language\csname l@#1\endcsname
4449   \edef\language\language{#1}%
4450   \bbl@hook@everylanguage{#1}%
4451   % > luatex
4452   \bbl@get@enc#1:.\@@@
4453   \begingroup
4454     \lefthyphenmin\m@ne
4455     \bbl@hook@loadpatterns{#2}%
4456     % > luatex
```

```

4457 \ifnum\lefthyphenmin=\m@ne
4458 \else
4459 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4460 \the\lefthyphenmin\the\righthyphenmin}%
4461 \fi
4462 \endgroup
4463 \def\bbl@tempa{#3}%
4464 \ifx\bbl@tempa@empty\else
4465 \bbl@hook@loadexceptions{#3}%
4466 % > luatex
4467 \fi
4468 \let\bbl@elt\relax
4469 \edef\bbl@languages{%
4470 \bbl@languages\bbl@elt{#1}\the\language}{#2}\bbl@tempa}%
4471 \ifnum\the\language=\z@
4472 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4473 \set@hyphenmins\tw@thr@@\relax
4474 \else
4475 \expandafter\expandafter\expandafter\set@hyphenmins
4476 \csname #1hyphenmins\endcsname
4477 \fi
4478 \the\toks@
4479 \toks@{}%
4480 \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.

```

4481 \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.

```

4482 \def\bbl@hook@everylanguage#1{}
4483 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4484 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4485 \def\bbl@hook@loadkernel#1{%
4486 \def\addlanguage{\csname newlanguage\endcsname}%
4487 \def\adddialect##1##2{%
4488 \global\chardef##1##2\relax
4489 \wlog{\string##1 = a dialect from \string\language##2}}%
4490 \def\iflanguage##1{%
4491 \expandafter\ifx\csname l@##1\endcsname\relax
4492 \@nolanerr{##1}%
4493 \else
4494 \ifnum\csname l@##1\endcsname=\language
4495 \expandafter\expandafter\expandafter\@firstoftwo
4496 \else
4497 \expandafter\expandafter\expandafter\@secondoftwo
4498 \fi
4499 \fi}%
4500 \def\providehyphenmins##1##2{%
4501 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4502 \@namedef{##1hyphenmins}{##2}%
4503 \fi}%
4504 \def\set@hyphenmins##1##2{%
4505 \lefthyphenmin##1\relax
4506 \righthyphenmin##2\relax}%
4507 \def\selectlanguage{%
4508 \errhelp{Selecting a language requires a package supporting it}%
4509 \errmessage{No multilingual package has been loaded}}%
4510 \let\foreignlanguage\selectlanguage
4511 \let\otherlanguage\selectlanguage

```

```

4512 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4513 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4514 \def\setlocale{%
4515   \errhelp{Find an armchair, sit down and wait}%
4516   \errmessage{(babel) Not yet available}}%
4517 \let\uselocale\setlocale
4518 \let\locale\setlocale
4519 \let\selectlocale\setlocale
4520 \let\localename\setlocale
4521 \let\textlocale\setlocale
4522 \let\textlanguage\setlocale
4523 \let\languagetext\setlocale}
4524 \begingroup
4525 \def\AddBabelHook#1#2{%
4526   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4527     \def\next{\toks1}%
4528     \else
4529       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4530     \fi
4531     \next}
4532 \ifx\directlua@undefined
4533   \ifx\XeTeXinputencoding@undefined\else
4534     \input xebabel.def
4535   \fi
4536 \else
4537   \input luababel.def
4538 \fi
4539 \openin1 = babel-\bbl@format.cfg
4540 \ifeof1
4541 \else
4542   \input babel-\bbl@format.cfg\relax
4543 \fi
4544 \closein1
4545 \endgroup
4546 \bbl@hook@loadkernel{switch.def}

```

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

```

4547 \openin1 = language.dat

```

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

```

4548 \def\language{english}%
4549 \ifeof1
4550 \message{I couldn't find the file language.dat,\space
4551   I will try the file hyphen.tex}
4552 \input hyphen.tex\relax
4553 \chardef\l@english\z@
4554 \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`.

```

4555 \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.

```

4556 \loop
4557   \endlinechar\m@ne
4558   \read1 to \bbl@line
4559   \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.

```
4560 \if T\ifeoflF\fi T\relax
4561 \ifx\bbl@line\empty\else
4562 \edef\bbl@line{\bbl@line\space\space\space}%
4563 \expandafter\process@line\bbl@line\relax
4564 \fi
4565 \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.

```
4566 \begingroup
4567 \def\bbl@elt#1#2#3#4{%
4568 \global\language=#2\relax
4569 \gdef\language#1}%
4570 \def\bbl@elt##1##2##3##4{}}%
4571 \bbl@languages
4572 \endgroup
4573 \fi
4574 \closeinl
```

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

```
4575 \if/\the\toks@/\else
4576 \errhelp{language.dat loads no language, only synonyms}
4577 \errmessage{Orphan language synonym}
4578 \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.

```
4579 \let\bbl@line\undefined
4580 \let\process@line\undefined
4581 \let\process@synonym\undefined
4582 \let\process@language\undefined
4583 \let\bbl@get@enc\undefined
4584 \let\bbl@hyph@enc\undefined
4585 \let\bbl@tempa\undefined
4586 \let\bbl@hook@loadkernel\undefined
4587 \let\bbl@hook@everylanguage\undefined
4588 \let\bbl@hook@loadpatterns\undefined
4589 \let\bbl@hook@loadexceptions\undefined
4590 </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`).

```
4591 <<*More package options>> ≡
4592 \chardef\bbl@bidimode\z@
4593 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4594 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4595 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4596 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4597 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4598 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4599 <</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. `\font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

```

4600 <<*Font selection>> ≡
4601 \bbl@trace{Font handling with fontspec}
4602 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4603 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4604 \DisableBabelHook{babel-fontspec}
4605 \@onlypreamble\babelfont
4606 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4607   \ifx\fontspec\undefined
4608     \usepackage{fontspec}%
4609     \fi
4610     \EnableBabelHook{babel-fontspec}%
4611     \edef\bbl@tempa{#1}%
4612     \def\bbl@tempb{#2}% Used by \bbl@bblfont
4613     \bbl@bblfont}
4614 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4615   \bbl@ifunset{\bbl@tempb family}%
4616     {\bbl@providefam{\bbl@tempb}}%
4617     {}%
4618   % For the default font, just in case:
4619   \bbl@ifunset{\bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{%}%
4620   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4621   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save \bbl@rmdflt@
4622     \bbl@exp{%
4623       \let<\bbl@tempb dflt@\languageName>\<\bbl@tempb dflt@>%
4624       \\bbl@font@set<\bbl@tempb dflt@\languageName>%
4625       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4626   {\bbl@foreach\bbl@tempa{% i.e., \bbl@rmdflt@lang / *sctr
4627     \bbl@csarg\def{\bbl@tempb dflt@##1}{<{#1}{#2}}}%

```

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

```

4628 \def\bbl@providefam#1{%
4629   \bbl@exp{%
4630     \\newcommand<\<#1default>{}% Just define it
4631     \\bbl@add@list\\bbl@font@fams{#1}%
4632     \\DeclareRobustCommand<\<#1family>{%
4633       \\not@math@alphabet<\<#1family>\relax
4634       % \\prepare@family@series@update{#1}<\<#1default>% TODO. Fails
4635       \\fontfamily<\<#1default>%
4636       \<\ifx>\\UseHooks\\@undefined<\else>\\UseHook{#1family}>\<\fi>%
4637       \\selectfont}%
4638       \\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.

```

4639 \def\bbl@nostdfont#1{%
4640   \bbl@ifunset{\bbl@WFF@\fontfamily}%
4641   {\bbl@csarg\gdef{WFF@\fontfamily}{% Flag, to avoid dupl warns
4642     \bbl@infowarn{The current font is not a babel standard family:\%
4643       #1%
4644       \fontname\font\\%
4645       There is nothing intrinsically wrong with this warning, and\\%
4646       you can ignore it altogether if you do not need these\\%
4647       families. But if they are used in the document, you should be\\%
4648       aware 'babel' will not set Script and Language for them, so\\%
4649       you may consider defining a new family with \string\babelfont.\\%
4650       See the manual for further details about \string\babelfont.\\%
4651       Reported}}
4652   }%
4653 \gdef\bbl@switchfont{%
4654   \bbl@ifunset{\bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{%}%

```

```

4655 \bbl@exp{% e.g., Arabic -> arabic
4656 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}}%
4657 \bbl@foreach\bbl@font@fams{%
4658 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4659 {\bbl@ifunset{\bbl@##1dflt*@\bbl@tempa}% (2) from script?
4660 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4661 {}% 123=F - nothing!
4662 {\bbl@exp{% 3=T - from generic
4663 \global\let<\bbl@##1dflt@\language>%
4664 \<\bbl@##1dflt@>}}}%
4665 {\bbl@exp{% 2=T - from script
4666 \global\let<\bbl@##1dflt@\language>%
4667 \<\bbl@##1dflt*@\bbl@tempa>}}}%
4668 {}}% 1=T - language, already defined
4669 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4670 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4671 \bbl@ifunset{\bbl@##1dflt@\language}%
4672 {\bbl@cs{famrst@##1}%
4673 \global\bbl@csarg\let{famrst@##1}\relax}%
4674 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4675 \\bbl@add\\originalTeX{%
4676 \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4677 \<##1default>\<##1family>{##1}}}%
4678 \\bbl@font@set<\bbl@##1dflt@\language>% the main part!
4679 \<##1default>\<##1family>}}}%
4680 \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`.

```

4681 \ifx\f@family\undefined\else % if latex
4682 \ifcase\bbl@engine % if pdftex
4683 \let\bbl@ckeckstdfonts\relax
4684 \else
4685 \def\bbl@ckeckstdfonts{%
4686 \begingroup
4687 \global\let\bbl@ckeckstdfonts\relax
4688 \let\bbl@tempa\empty
4689 \bbl@foreach\bbl@font@fams{%
4690 \bbl@ifunset{\bbl@##1dflt@}%
4691 {\@nameuse{##1family}%
4692 \bbl@csarg\gdef{WFF@f@family}{}% Flag
4693 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4694 \space\space\fontname\font\\}%
4695 \bbl@csarg\xdef{##1dflt@}{f@family}%
4696 \expandafter\xdef\csname ##1default\endcsname{f@family}}}%
4697 {}}%
4698 \ifx\bbl@tempa\empty\else
4699 \bbl@info{The following font families will use the default\\%
4700 settings for all or some languages:\\%
4701 \bbl@tempa
4702 There is nothing intrinsically wrong with it, but\\%
4703 'babel' will no set Script and Language, which could\\%
4704 be relevant in some languages. If your document uses\\%
4705 these families, consider redefining them with \string\babelfont.\\%
4706 Reported}%
4707 \fi
4708 \endgroup}
4709 \fi
4710 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4711 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4712 \bbl@xin@{<>}{#1}%
4713 \ifin@
4714 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4715 \fi
4716 \bbl@exp{% 'Unprotected' macros return prev values
4717 \def\\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
4718 \\bbl@ifsamestring{#2}{\f@family}%
4719 {\#3%
4720 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4721 \let\\bbl@tempa\relax}%
4722 {}}}
```

Loaded locally, which does its job, but very must be global. The problem is how.

```

4723 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4724 \let\bbl@tempe\bbl@mapselect
4725 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4726 \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}{}}%
4727 \let\bbl@mapselect\relax
4728 \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
4729 \let#4@empty % Make sure \renewfontfamily is valid
4730 \bbl@set@renderer
4731 \bbl@exp{%
4732 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% e.g., '\rmfamily '
4733 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4734 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4735 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4736 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4737 \\renewfontfamily\\#4%
4738 [\bbl@cl{lsys},% xetex removes unknown features :-(
4739 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4740 #2]}{#3}% i.e., \bbl@exp{..}{#3}
4741 \bbl@unset@renderer
4742 \begingroup
4743 #4%
4744 \xdef#1{\f@family}% e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4745 \endgroup % TODO. Find better tests:
4746 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4747 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4748 \ifin@
4749 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4750 \fi
4751 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4752 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4753 \ifin@
4754 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4755 \fi
4756 \let#4\bbl@temp@fam
4757 \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4758 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4759 \def\bbl@font@rst#1#2#3#4{%
4760 \bbl@ccarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}
```

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


```

4761 \def\bbl@font@fams{rm,sf,tt}
4762 <</Font selection>>

```

\BabelFootnote Footnotes.

```

4763 <<*Footnote changes>> ≡
4764 \bbl@trace{Bidi footnotes}
4765 \ifnum\bbl@bidimode>\z@ % Any bidi=
4766 \def\bbl@footnote#1#2#3{%
4767   \@ifnextchar[%
4768     {\bbl@footnote@o{#1}{#2}{#3}}%
4769     {\bbl@footnote@x{#1}{#2}{#3}}}
4770 \long\def\bbl@footnote@x#1#2#3#4{%
4771   \bgroup
4772     \select@language@x{\bbl@main@language}%
4773     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4774   \egroup}
4775 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4776   \bgroup
4777     \select@language@x{\bbl@main@language}%
4778     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4779   \egroup}
4780 \def\bbl@footnotetext#1#2#3{%
4781   \@ifnextchar[%
4782     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4783     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4784 \long\def\bbl@footnotetext@x#1#2#3#4{%
4785   \bgroup
4786     \select@language@x{\bbl@main@language}%
4787     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4788   \egroup}
4789 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4790   \bgroup
4791     \select@language@x{\bbl@main@language}%
4792     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4793   \egroup}
4794 \def\BabelFootnote#1#2#3#4{%
4795   \ifx\bbl@fn@footnote@undefined
4796     \let\bbl@fn@footnote\footnote
4797   \fi
4798   \ifx\bbl@fn@footnotetext@undefined
4799     \let\bbl@fn@footnotetext\footnotetext
4800   \fi
4801   \bbl@ifblank{#2}%
4802     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4803     \@namedef{\bbl@stripslash#1text}%
4804     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4805     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4806     \@namedef{\bbl@stripslash#1text}%
4807     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4808 \fi
4809 <</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.

```

4810 <<*xetex>
4811 \def\BabelStringsDefault{unicode}

```

```

4812 \let\xebbl@stop\relax
4813 \AddBabelHook{xetex}{encodedcommands}{%
4814   \def\bbl@tempa{#1}%
4815   \ifx\bbl@tempa\@empty
4816     \XeTeXinputencoding"bytes"%
4817   \else
4818     \XeTeXinputencoding"#1"%
4819   \fi
4820   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4821 \AddBabelHook{xetex}{stopcommands}{%
4822   \xebbl@stop
4823   \let\xebbl@stop\relax}
4824 \def\bbl@input@classes{% Used in CJK intraspaces
4825   \input{load-unicode-xetex-classes.tex}%
4826   \let\bbl@input@classes\relax}
4827 \def\bbl@intraspace#1 #2 #3\@{%
4828   \bbl@csarg\gdef{xeisp@\language}%
4829   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4830 \def\bbl@intrapenalty#1\@{%
4831   \bbl@csarg\gdef{xeipn@\language}%
4832   {\XeTeXlinebreakpenalty #1\relax}}
4833 \def\bbl@provide@intraspace{%
4834   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4835   \ifin\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4836   \ifin@
4837     \bbl@ifunset{bbl@intsp@\language}{}%
4838     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4839       \ifx\bbl@KVP@intraspace\@nnil
4840         \bbl@exp{%
4841           \\bbl@intraspace\bbl@cl{intsp}\\@}%
4842         \fi
4843         \ifx\bbl@KVP@intrapenalty\@nnil
4844           \bbl@intrapenalty0\@@
4845         \fi
4846         \fi
4847         \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4848           \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4849         \fi
4850         \ifx\bbl@KVP@intrapenalty\@nnil\else
4851           \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4852         \fi
4853         \bbl@exp{%
4854           % TODO. Execute only once (but redundant):
4855           \\bbl@add<extras\language>{%
4856             \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4857             \<bbl@xeisp@\language>%
4858             \<bbl@xeipn@\language>}%
4859           \\bbl@tglobal\<extras\language>%
4860           \\bbl@add<noextras\language>{%
4861             \XeTeXlinebreaklocale ""}%
4862           \\bbl@tglobal\<noextras\language>}%
4863           \ifx\bbl@ispacesize\@undefined
4864             \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4865             \ifx\AtBeginDocument\@notprerr
4866               \expandafter\@secondoftwo % to execute right now
4867             \fi
4868             \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4869           \fi}%
4870   \fi}
4871 \ifx\DisableBabelHook\@undefined\endinput\fi %%% TODO: why
4872 \let\bbl@set@renderer\relax
4873 \let\bbl@unset@renderer\relax
4874 <@Font selection>

```

```
4875 \def\bbl@provide@extra#1{}
```

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.

```
4876 \ifnum\xe@alloc@intercharclass<\thr@@
4877 \xe@alloc@intercharclass\thr@@
4878 \fi
4879 \chardef\bbl@xe@class@default=\z@
4880 \chardef\bbl@xe@class@cjkideogram=\@ne
4881 \chardef\bbl@xe@class@cjkleftpunctuation=\tw@
4882 \chardef\bbl@xe@class@cjkrightpunctuation=\thr@@
4883 \chardef\bbl@xe@class@boundary=4095
4884 \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.

```
4885 \AddBabelHook{babel-interchar}{beforeextras}{%
4886 \nameuse{bbl@xechars@language}
4887 \DisableBabelHook{babel-interchar}
4888 \protected\def\bbl@charclass#1{%
4889 \ifnum\count@<\z@
4890 \count@-\count@
4891 \loop
4892 \bbl@exp{%
4893 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4894 \XeTeXcharclass\count@ \bbl@tempc
4895 \ifnum\count@<`#1\relax
4896 \advance\count@\@ne
4897 \repeat
4898 \else
4899 \babel@savevariable{\XeTeXcharclass`#1}%
4900 \XeTeXcharclass`#1 \bbl@tempc
4901 \fi
4902 \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.

```
4903 \newcommand\bbl@ifinterchar[1]{%
4904 \let\bbl@tempa\@gobble % Assume to ignore
4905 \edef\bbl@tempb{\zap@space#1 \@empty}%
4906 \ifx\bbl@KVP@interchar\@nnil\else
4907 \bbl@replace\bbl@KVP@interchar{ }{,}%
4908 \bbl@foreach\bbl@tempb{%
4909 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
4910 \ifin@
4911 \let\bbl@tempa\@firstofone
4912 \fi}%
4913 \fi
4914 \bbl@tempa}
4915 \newcommand\IfBabelIntercharT[2]{%
4916 \bbl@carg\bbl@add{bbl@icsave\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
4917 \newcommand\babelcharclass[3]{%
4918 \EnableBabelHook{babel-interchar}%
4919 \bbl@carg\newXeTeXintercharclass{xe@class@#2@#1}%
4920 \def\bbl@tempb##1{%
```

```

4921 \ifx##1\@empty\else
4922 \ifx##1-%
4923 \bbl@upto
4924 \else
4925 \bbl@class{%-
4926 \ifcat\@noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4927 \fi
4928 \expandafter\bbl@tempb
4929 \fi}%
4930 \bbl@ifunset{\bbl@xechars@#1}%
4931 {\toks@{%-
4932 \babel@savevariable\XeTeXinterchartokenstate
4933 \XeTeXinterchartokenstate\@ne
4934 }}%
4935 {\toks@\expandafter\expandafter\expandafter{%-
4936 \csname bbl@xechars@#1\endcsname}}%
4937 \bbl@csarg\edef{\xechars@#1}{%-
4938 \the\toks@
4939 \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
4940 \bbl@tempb#3\@empty}}
4941 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
4942 \protected\def\bbl@upto{%-
4943 \ifnum\count@>\z@
4944 \advance\count@\@ne
4945 \count@\count@
4946 \else\ifnum\count@=\z@
4947 \bbl@class{-}%
4948 \else
4949 \bbl@error{double-hyphens-class}{\count@}{\count@}%
4950 \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>`.

```

4951 \def\bbl@ignoreinterchar{%-
4952 \ifnum\language=\l@nohyphenation
4953 \expandafter\@gobble
4954 \else
4955 \expandafter\@firstofone
4956 \fi}
4957 \newcommand\babelinterchar[5][{}]{%-
4958 \let\bbl@kv@label\@empty
4959 \bbl@forkv{#1}{\bbl@csarg\edef{\kv@##1}{##2}}%
4960 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4961 {\bbl@ignoreinterchar{#5}}%
4962 \bbl@csarg\let{\ic@\bbl@kv@label @#2}\@firstofone
4963 \bbl@expf{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%-
4964 \bbl@expf{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%-
4965 \XeTeXinterchartoks
4966 \@nameuse{\bbl@xeclasse@\bbl@tempa @#2}%
4967 \bbl@ifunset{\bbl@xeclasse@\bbl@tempa @#2}{\bbl@tempa @#2}%
4968 \@nameuse{\bbl@xeclasse@\bbl@tempb @#2}%
4969 \bbl@ifunset{\bbl@xeclasse@\bbl@tempb @#2}{\bbl@tempb @#2}%
4970 = \expandafter{%-
4971 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
4972 \csname\zap@space bbl@xeinter@\bbl@kv@label
4973 @#3@#4@#2 \@empty\endcsname}}}}
4974 \DeclareRobustCommand\enablelocaleinterchar[1]{%-
4975 \bbl@ifunset{\bbl@ic@#1@language}%
4976 {\bbl@error{unknown-interchar}{#1}{\count@}}%
4977 {\bbl@csarg\let{\ic@#1@language}\@firstofone}}
4978 \DeclareRobustCommand\disablelocaleinterchar[1]{%-
4979 \bbl@ifunset{\bbl@ic@#1@language}%

```

```

4980     {\bbl@error{unknown-interchar-b}{#1}{}}}%
4981     {\bbl@csarg\let{ic@#1@\language}\@gobble}}
4982 </xetex>

```

10.3. Layout

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

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

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

```

4983 <*xetex | texxet>
4984 \providecommand\bbl@provide@intraspace{}
4985 \bbl@trace{Redefinitions for bidi layout}
4986 \def\bbl@sspre@caption{% TODO: Unused!
4987   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4988 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4989 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4990 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4991 \ifnum\bbl@bidimode>\z@ % TODO: always?
4992   \def\@hangfrom#1{%
4993     \setbox\@tempboxa\hbox{#1}%
4994     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4995     \noindent\box\@tempboxa}
4996 \def\raggedright{%
4997   \let\@centercr
4998   \bbl@startskip\z@skip
4999   \@rightskip\@flushglue
5000   \bbl@endskip\@rightskip
5001   \parindent\z@
5002   \parfillskip\bbl@startskip}
5003 \def\raggedleft{%
5004   \let\@centercr
5005   \bbl@startskip\@flushglue
5006   \bbl@endskip\z@skip
5007   \parindent\z@
5008   \parfillskip\bbl@endskip}
5009 \fi
5010 \IfBabelLayout{lists}
5011   {\bbl@sreplace\list
5012     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5013     \def\bbl@listleftmargin{%
5014       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5015     \ifcase\bbl@engine
5016       \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
5017       \def\p@enumiii{\p@enumii}\theenumii{}%
5018     \fi
5019     \bbl@sreplace\@verbatim
5020     {\leftskip\@totalleftmargin}%
5021     {\bbl@startskip\textwidth
5022       \advance\bbl@startskip-\linewidth}%
5023     \bbl@sreplace\@verbatim
5024     {\rightskip\z@skip}%
5025     {\bbl@endskip\z@skip}}%
5026   {}
5027 \IfBabelLayout{contents}
5028   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5029     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5030   {}
5031 \IfBabelLayout{columns}
5032   {\bbl@sreplace\@outputdblcol{\hb@xt\textwidth}{\bbl@outputbox}%

```

```

5033 \def\bbl@outputbox#1{%
5034 \hb@xt@\textwidth{%
5035 \hskip\columnwidth
5036 \hfil
5037 {\normalcolor\vrule \@width\columnseprule}%
5038 \hfil
5039 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5040 \hskip-\textwidth
5041 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5042 \hskip\columnsep
5043 \hskip\columnwidth}}}%
5044 {}
5045 <@Footnote changes@>
5046 \IfBabelLayout{footnotes}%
5047 {\BabelFootnote\footnote\language\language{}{}}%
5048 \BabelFootnote\localfootnote\language\language{}{}}%
5049 \BabelFootnote\mainfootnote{}{}}%
5050 {}

```

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.

```

5051 \IfBabelLayout{counters*}%
5052 {\bbl@add\bbl@opt@layout{.counters.}%
5053 \AddToHook{shipout/before}{%
5054 \let\bbl@tempa\babelsublr
5055 \let\babelsublr\@firstofone
5056 \let\bbl@save@thepage\thepage
5057 \protected@edef\thepage{\thepage}%
5058 \let\babelsublr\bbl@tempa}%
5059 \AddToHook{shipout/after}{%
5060 \let\thepage\bbl@save@thepage}}}%
5061 \IfBabelLayout{counters}%
5062 {\let\bbl@latinarabic=\@arabic
5063 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5064 \let\bbl@asciroman=\@roman
5065 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5066 \let\bbl@asciiRoman=\@Roman
5067 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
5068 \fi % end if layout
5069 </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).

```

5070 <*texxet>
5071 \def\bbl@provide@extra#1{%
5072 % == auto-select encoding ==
5073 \ifx\bbl@encoding@select@off\@empty\else
5074 \bbl@ifunset{\bbl@encoding@#1}%
5075 {\def\@elt##1{,##1,}%
5076 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5077 \count@\z@
5078 \bbl@foreach\bbl@tempe{%
5079 \def\bbl@tempd{##1}% Save last declared
5080 \advance\count@\@ne}%
5081 \ifnum\count@>\@ne % (1)
5082 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5083 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5084 \bbl@replace\bbl@tempa{ },}%
5085 \global\bbl@csarg\let{encoding@#1}\@empty
5086 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5087 \ifin@ \else % if main encoding included in ini, do nothing

```

```

5088      \let\bbl@tempb\relax
5089      \bbl@foreach\bbl@tempa{%
5090        \ifx\bbl@tempb\relax
5091          \bbl@xin@{,##1,}{,\bbl@tempe,}%
5092          \ifin@def\bbl@tempb{##1}\fi
5093        \fi}%
5094      \ifx\bbl@tempb\relax\else
5095        \bbl@exp{%
5096          \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5097          \gdef\<bbl@encoding@#1>{%
5098            \\babel@save\\f@encoding
5099            \\bbl@add\\originalTeX{\\selectfont}%
5100            \\fontencoding\bbl@tempb}%
5101            \\selectfont}}%
5102        \fi
5103      \fi
5104    \fi}%
5105  }%
5106 \fi}
5107 </texxet>

```

10.5. LuaTeX

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

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

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

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

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

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

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

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

```

5108 <*\luatex>
5109 \directlua{ Babel = Babel or {} } % DL2
5110 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5111 \bbl@trace{Read language.dat}
5112 \ifx\bbl@readstream\undefined
5113   \csname newread\endcsname\bbl@readstream
5114 \fi
5115 \begingroup

```

```

5116 \toks@{}
5117 \count@\z@ % 0=start, 1=0th, 2=normal
5118 \def\bbl@process@line#1#2 #3 #4 {%
5119   \ifx=#1%
5120     \bbl@process@synonym{#2}%
5121   \else
5122     \bbl@process@language{#1#2}{#3}{#4}%
5123   \fi
5124   \ignorespaces}
5125 \def\bbl@manylang{%
5126   \ifnum\bbl@last>\@ne
5127     \bbl@info{Non-standard hyphenation setup}%
5128   \fi
5129   \let\bbl@manylang\relax}
5130 \def\bbl@process@language#1#2#3{%
5131   \ifcase\count@
5132     \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5133   \or
5134     \count@\tw@
5135   \fi
5136   \ifnum\count@=\tw@
5137     \expandafter\addlanguage\csname l@#1\endcsname
5138     \language\allocationnumber
5139     \chardef\bbl@last\allocationnumber
5140     \bbl@manylang
5141     \let\bbl@elt\relax
5142     \xdef\bbl@languages{%
5143       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5144   \fi
5145   \the\toks@
5146   \toks@{}}
5147 \def\bbl@process@synonym@aux#1#2{%
5148   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5149   \let\bbl@elt\relax
5150   \xdef\bbl@languages{%
5151     \bbl@languages\bbl@elt{#1}{#2}{}}}%
5152 \def\bbl@process@synonym#1{%
5153   \ifcase\count@
5154     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5155   \or
5156     \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5157   \else
5158     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5159   \fi}
5160 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5161   \chardef\l@english\z@
5162   \chardef\l@USenglish\z@
5163   \chardef\bbl@last\z@
5164   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5165   \gdef\bbl@languages{%
5166     \bbl@elt{english}{0}{hyphen.tex}}%
5167     \bbl@elt{USenglish}{0}{}}
5168 \else
5169   \global\let\bbl@languages@format\bbl@languages
5170   \def\bbl@elt#1#2#3#4{% Remove all except language 0
5171     \ifnum#2>\z@\else
5172       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5173     \fi}%
5174   \xdef\bbl@languages{\bbl@languages}%
5175   \fi
5176   \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5177   \bbl@languages
5178   \openin\bbl@readstream=language.dat

```



```

5179 \ifeof\bbl@readstream
5180 \bbl@warning{I couldn't find language.dat. No additional\\%
5181 patterns loaded. Reported}%
5182 \else
5183 \loop
5184 \endlinechar\m@ne
5185 \read\bbl@readstream to \bbl@line
5186 \endlinechar\^^M
5187 \if T\ifeof\bbl@readstream F\fi T\relax
5188 \ifx\bbl@line\@empty\else
5189 \edef\bbl@line{\bbl@line\space\space\space}%
5190 \expandafter\bbl@process@line\bbl@line\relax
5191 \fi
5192 \repeat
5193 \fi
5194 \closein\bbl@readstream
5195 \endgroup
5196 \bbl@trace{Macros for reading patterns files}
5197 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
5198 \ifx\babelcatcodetablenum\@undefined
5199 \ifx\newcatcodetable\@undefined
5200 \def\babelcatcodetablenum{5211}
5201 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5202 \else
5203 \newcatcodetable\babelcatcodetablenum
5204 \newcatcodetable\bbl@pattcodes
5205 \fi
5206 \else
5207 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5208 \fi
5209 \def\bbl@luapatterns#1#2{%
5210 \bbl@get@enc#1:.\@@@
5211 \setbox\z@\hbox\bgroup
5212 \beginingroup
5213 \savecatcodetable\babelcatcodetablenum\relax
5214 \initcatcodetable\bbl@pattcodes\relax
5215 \catcodetable\bbl@pattcodes\relax
5216 \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5217 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5218 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5219 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5220 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5221 \catcode\`=12 \catcode\'=12 \catcode\"=12
5222 \input #1\relax
5223 \catcodetable\babelcatcodetablenum\relax
5224 \endgroup
5225 \def\bbl@tempa{#2}%
5226 \ifx\bbl@tempa\@empty\else
5227 \input #2\relax
5228 \fi
5229 \egroup}%
5230 \def\bbl@patterns@lua#1{%
5231 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5232 \csname l@#1\endcsname
5233 \edef\bbl@tempa{#1}%
5234 \else
5235 \csname l@#1:f@encoding\endcsname
5236 \edef\bbl@tempa{#1:f@encoding}%
5237 \fi\relax
5238 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5239 \@ifundefined{bbl@hyphendata@the\language}%
5240 {\def\bbl@elt##1##2##3##4{%
5241 \ifnum##2=\csname l@#1:f@encoding\endcsname % #2=spanish, dutch:OT1...

```

```

5242     \def\bbl@tempb{##3}%
5243     \ifx\bbl@tempb\@empty\else % if not a synonymous
5244     \def\bbl@tempc{##3}{##4}%
5245     \fi
5246     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5247     \fi}%
5248     \bbl@languages
5249     \@ifundefined{bbl@hyphendata@the\language}%
5250     {\bbl@info{No hyphenation patterns were set for\%
5251     language '\bbl@tempa'. Reported}}%
5252     {\expandafter\expandafter\expandafter\bbl@luapatterns
5253     \csname bbl@hyphendata@the\language\endcsname}}{}%
5254 \endinput\fi

```

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

```

5255 \ifx\DisableBabelHook\@undefined
5256 \AddBabelHook{luatex}{everylanguage}{%
5257 \def\process@language##1##2##3{%
5258 \def\process@line####1####2 ####3 ####4 {}}
5259 \AddBabelHook{luatex}{loadpatterns}{%
5260 \input #1\relax
5261 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5262 {{#1}}}%
5263 \AddBabelHook{luatex}{loadexceptions}{%
5264 \input #1\relax
5265 \def\bbl@tempb##1##2{{##1}{##1}}%
5266 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5267 {\expandafter\expandafter\expandafter\bbl@tempb
5268 \csname bbl@hyphendata@the\language\endcsname}}
5269 \endinput\fi

```

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

```

5270 \beginingroup % TODO - to a lua file % DL3
5271 \catcode`\%=12
5272 \catcode`\'=12
5273 \catcode`\|=12
5274 \catcode`\:=12
5275 \directlua{
5276 Babel.locale_props = Babel.locale_props or {}
5277 function Babel.lua_error(e, a)
5278 tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5279 e .. '}{' .. (a or '') .. '}{}{}')
5280 end
5281 function Babel.bytes(line)
5282 return line:gsub("(.)",
5283 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5284 end
5285 function Babel.begin_process_input()
5286 if luatexbase and luatexbase.add_to_callback then
5287 luatexbase.add_to_callback('process_input_buffer',
5288 Babel.bytes, 'Babel.bytes')
5289 else
5290 Babel.callback = callback.find('process_input_buffer')
5291 callback.register('process_input_buffer', Babel.bytes)
5292 end
5293 end
5294 function Babel.end_process_input ()
5295 if luatexbase and luatexbase.remove_from_callback then
5296 luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5297 else
5298 callback.register('process_input_buffer', Babel.callback)
5299 end
5300 end

```

```

5301 function Babel.str_to_nodes(fn, matches, base)
5302   local n, head, last
5303   if fn == nil then return nil end
5304   for s in string.utfvalues(fn(matches)) do
5305     if base.id == 7 then
5306       base = base.replace
5307     end
5308     n = node.copy(base)
5309     n.char = s
5310     if not head then
5311       head = n
5312     else
5313       last.next = n
5314     end
5315     last = n
5316   end
5317   return head
5318 end
5319 Babel.linebreaking = Babel.linebreaking or {}
5320 Babel.linebreaking.before = {}
5321 Babel.linebreaking.after = {}
5322 Babel.locale = {}
5323 function Babel.linebreaking.add_before(func, pos)
5324   tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5325   if pos == nil then
5326     table.insert(Babel.linebreaking.before, func)
5327   else
5328     table.insert(Babel.linebreaking.before, pos, func)
5329   end
5330 end
5331 function Babel.linebreaking.add_after(func)
5332   tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5333   table.insert(Babel.linebreaking.after, func)
5334 end
5335 function Babel.addpatterns(pp, lg)
5336   local lg = lang.new(lg)
5337   local pats = lang.patterns(lg) or ''
5338   lang.clear_patterns(lg)
5339   for p in pp:gmatch('^%s+') do
5340     ss = ''
5341     for i in string.utfcharacters(p:gsub('%d', '')) do
5342       ss = ss .. '%d?' .. i
5343     end
5344     ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5345     ss = ss:gsub('%%.%d%?$', '%%.')
5346     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5347     if n == 0 then
5348       tex.sprint(
5349         [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5350         .. p .. [[]])
5351       pats = pats .. ' ' .. p
5352     else
5353       tex.sprint(
5354         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5355         .. p .. [[]])
5356     end
5357   end
5358   lang.patterns(lg, pats)
5359 end
5360 Babel.characters = Babel.characters or {}
5361 Babel.ranges = Babel.ranges or {}
5362 function Babel.hlist_has_bidi(head)
5363   local has_bidi = false

```

```

5364     local ranges = Babel.ranges
5365     for item in node.traverse(head) do
5366         if item.id == node.id'glyph' then
5367             local itemchar = item.char
5368             local chardata = Babel.characters[itemchar]
5369             local dir = chardata and chardata.d or nil
5370             if not dir then
5371                 for nn, et in ipairs(ranges) do
5372                     if itemchar < et[1] then
5373                         break
5374                     elseif itemchar <= et[2] then
5375                         dir = et[3]
5376                         break
5377                     end
5378                 end
5379             end
5380             if dir and (dir == 'al' or dir == 'r') then
5381                 has_bidi = true
5382             end
5383         end
5384     end
5385     return has_bidi
5386 end
5387 function Babel.set_chranges_b (script, chrng)
5388     if chrng == '' then return end
5389     texio.write('Replacing ' .. script .. ' script ranges')
5390     Babel.script_blocks[script] = {}
5391     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.%)s') do
5392         table.insert(
5393             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5394     end
5395 end
5396 function Babel.discard_sublr(str)
5397     if str:find( [[\string\indexentry]] ) and
5398        str:find( [[\string\babelsublr]] ) then
5399         str = str:gsub( [[\string\babelsublr%s*{%b{}}]],
5400             function(m) return m:sub(2,-2) end )
5401     end
5402     return str
5403 end
5404 }
5405 \endgroup
5406 \ifx\newattribute\undefined\else % Test for plain
5407 \newattribute\bbl@attr@locale % DL4
5408 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5409 \AddBabelHook{luatex}{beforeextras}{%
5410     \setattribute\bbl@attr@locale\localeid}
5411 \fi
5412 \def\BabelStringsDefault{unicode}
5413 \let\luabbl@stop\relax
5414 \AddBabelHook{luatex}{encodedcommands}{%
5415     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5416     \ifx\bbl@tempa\bbl@tempb\else
5417         \directlua{Babel.begin_process_input()}%
5418         \def\luabbl@stop{%
5419             \directlua{Babel.end_process_input()}}%
5420     \fi}%
5421 \AddBabelHook{luatex}{stopcommands}{%
5422     \luabbl@stop
5423     \let\luabbl@stop\relax}
5424 \AddBabelHook{luatex}{patterns}{%
5425     \@ifundefined{bbl@hyphendata@the\language}%
5426     {\def\bbl@elt##1##2##3##4{%

```

```

5427 \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:0T1...
5428 \def\bbl@tempb{##3}%
5429 \ifx\bbl@tempb\empty\else % if not a synonymous
5430 \def\bbl@tempc{##3}{##4}%
5431 \fi
5432 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5433 \fi}%
5434 \bbl@languages
5435 \@ifundefined{bbl@hyphendata@the\language}%
5436 {\bbl@info{No hyphenation patterns were set for\%
5437 language '#2'. Reported}}%
5438 {\expandafter\expandafter\expandafter\bbl@luapatterns
5439 \csname bbl@hyphendata@the\language\endcsname}}}%
5440 \@ifundefined{bbl@patterns@}{}%
5441 \beginingroup
5442 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5443 \ifin@else
5444 \ifx\bbl@patterns@\empty\else
5445 \directlua{ Babel.addpatterns(
5446 [[\bbl@patterns@]], \number\language) }%
5447 \fi
5448 \@ifundefined{bbl@patterns@#1}%
5449 \empty
5450 {\directlua{ Babel.addpatterns(
5451 [[\space\csname bbl@patterns@#1\endcsname]],
5452 \number\language) }}%
5453 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5454 \fi
5455 \endgroup}%
5456 \bbl@exp{%
5457 \bbl@ifunset{bbl@prehc@language\name}{}%
5458 {\bbl@ifblank{\bbl@cs{prehc@language\name}}{}}%
5459 {\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.

```

5460 \@onlypreamble\babelpatterns
5461 \AtEndOfPackage{%
5462 \newcommand\babelpatterns[2][\empty]{%
5463 \ifx\bbl@patterns@\relax
5464 \let\bbl@patterns@\empty
5465 \fi
5466 \ifx\bbl@pttnlist@\empty\else
5467 \bbl@warning{%
5468 You must not intermingle \string\selectlanguage\space and\%
5469 \string\babelpatterns\space or some patterns will not\%
5470 be taken into account. Reported}%
5471 \fi
5472 \ifx@\empty#1%
5473 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5474 \else
5475 \edef\bbl@tempb{\zap@space#1 \empty}%
5476 \bbl@for\bbl@tempa\bbl@tempb{%
5477 \bbl@fixname\bbl@tempa
5478 \bbl@iflanguage\bbl@tempa{%
5479 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5480 \@ifundefined{bbl@patterns@\bbl@tempa}%
5481 \empty
5482 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5483 #2}}}%
5484 \fi}}

```

10.6. Southeast Asian scripts

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

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

```
5485 \def\bbl@intraspace#1 #2 #3\@@{%
5486   \directlua{
5487     Babel.intraspaces = Babel.intraspaces or {}
5488     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5489       {b = #1, p = #2, m = #3}
5490     Babel.locale_props[\the\localeid].intraspace = %
5491       {b = #1, p = #2, m = #3}
5492   }}
5493 \def\bbl@intrapenalty#1\@@{%
5494   \directlua{
5495     Babel.intrapenalties = Babel.intrapenalties or {}
5496     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5497     Babel.locale_props[\the\localeid].intrapenalty = #1
5498   }}
5499 \begingroup
5500 \catcode`\%=12
5501 \catcode`\&=14
5502 \catcode`\'=12
5503 \catcode`\-=12
5504 \gdef\bbl@seaintraspace{&
5505   \let\bbl@seaintraspace\relax
5506   \directlua{
5507     Babel.sea_enabled = true
5508     Babel.sea_ranges = Babel.sea_ranges or {}
5509     function Babel.set_chranges (script, chrng)
5510       local c = 0
5511       for s, e in string.gmatch(chrng..' ', '(.-%.%.(.-%)s') do
5512         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5513         c = c + 1
5514       end
5515     end
5516     function Babel.sea_disc_to_space (head)
5517       local sea_ranges = Babel.sea_ranges
5518       local last_char = nil
5519       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5520       for item in node.traverse(head) do
5521         local i = item.id
5522         if i == node.id'glyph' then
5523           last_char = item
5524         elseif i == 7 and item.subtype == 3 and last_char
5525           and last_char.char > 0x0C99 then
5526           quad = font.getfont(last_char.font).size
5527           for lg, rg in pairs(sea_ranges) do
5528             if last_char.char > rg[1] and last_char.char < rg[2] then
5529               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril1
5530               local intraspace = Babel.intraspaces[lg]
5531               local intrapenalty = Babel.intrapenalties[lg]
5532               local n
5533               if intrapenalty ~= 0 then
5534                 n = node.new(14, 0)      &% penalty
5535                 n.penalty = intrapenalty
5536                 node.insert_before(head, item, n)
5537               end
5538               n = node.new(12, 13)      &% (glue, spaceskip)
5539               node.setglue(n, intraspace.b * quad,
5540                 intraspace.p * quad,
5541                 intraspace.m * quad)
```

```

5542         node.insert_before(head, item, n)
5543         node.remove(head, item)
5544     end
5545 end
5546 end
5547 end
5548 end
5549 }&
5550 \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.

```

5551 \catcode`\%=14
5552 \gdef\bbl@cjkintraspacespace{%
5553   \let\bbl@cjkintraspacespace\relax
5554   \directlua{
5555     require('babel-data-cjk.lua')
5556     Babel.cjk_enabled = true
5557     function Babel.cjk_linebreak(head)
5558       local GLYPH = node.id'glyph'
5559       local last_char = nil
5560       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5561       local last_class = nil
5562       local last_lang = nil
5563
5564       for item in node.traverse(head) do
5565         if item.id == GLYPH then
5566
5567           local lang = item.lang
5568
5569           local LOCALE = node.get_attribute(item,
5570             Babel.attr_locale)
5571           local props = Babel.locale_props[LOCALE]
5572
5573           local class = Babel.cjk_class[item.char].c
5574
5575           if props.cjk_quotes and props.cjk_quotes[item.char] then
5576             class = props.cjk_quotes[item.char]
5577           end
5578
5579           if class == 'cp' then class = 'cl' % )) as CL
5580           elseif class == 'id' then class = 'I'
5581           elseif class == 'cj' then class = 'I' % loose
5582           end
5583
5584           local br = 0
5585           if class and last_class and Babel.cjk_breaks[last_class][class] then
5586             br = Babel.cjk_breaks[last_class][class]
5587           end
5588
5589           if br == 1 and props.linebreak == 'c' and
5590             lang ~= \the\l@nohyphenation\space and
5591             last_lang ~= \the\l@nohyphenation then
5592             local intrapenalty = props.intrapenalty
5593             if intrapenalty ~= 0 then
5594               local n = node.new(14, 0)      % penalty
5595               n.penalty = intrapenalty

```

```

5596         node.insert_before(head, item, n)
5597     end
5598     local intraspace = props.intraspace
5599     local n = node.new(12, 13)      % (glue, spaceskip)
5600     node.setglue(n, intraspace.b * quad,
5601                 intraspace.p * quad,
5602                 intraspace.m * quad)
5603     node.insert_before(head, item, n)
5604 end
5605
5606 if font.getfont(item.font) then
5607     quad = font.getfont(item.font).size
5608 end
5609 last_class = class
5610 last_lang = lang
5611 else % if penalty, glue or anything else
5612     last_class = nil
5613 end
5614 end
5615 lang.hyphenate(head)
5616 end
5617 }%
5618 \bbl@luahyphenate}
5619 \gdef\bbl@luahyphenate{%
5620 \let\bbl@luahyphenate\relax
5621 \directlua{
5622     luatexbase.add_to_callback('hyphenate',
5623     function (head, tail)
5624         if Babel.linebreaking.before then
5625             for k, func in ipairs(Babel.linebreaking.before) do
5626                 func(head)
5627             end
5628         end
5629         lang.hyphenate(head)
5630         if Babel.cjk_enabled then
5631             Babel.cjk_linebreak(head)
5632         end
5633         if Babel.linebreaking.after then
5634             for k, func in ipairs(Babel.linebreaking.after) do
5635                 func(head)
5636             end
5637         end
5638         if Babel.set_hboxed then
5639             Babel.set_hboxed(head)
5640         end
5641         if Babel.sea_enabled then
5642             Babel.sea_disc_to_space(head)
5643         end
5644     end,
5645     'Babel.hyphenate')
5646 }
5647 }
5648 \endgroup
5649 \def\bbl@provide@intraspace{%
5650 \bbl@ifunset\bbl@intsp@{language}{}%
5651 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\empty\else
5652 \bbl@xin@{/c}{/bbl@cl{lnbrk}}%
5653 \ifin@           % cjk
5654 \bbl@cjk@intraspace
5655 \directlua{
5656     Babel.locale_props = Babel.locale_props or {}
5657     Babel.locale_props[\the\localeid].linebreak = 'c'
5658 }%

```



```

5659      \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@@}%
5660      \ifx\bbl@KVP@intrapenalty\@nnil
5661        \bbl@intrapenalty0\@@
5662      \fi
5663    \else      % sea
5664      \bbl@seaintraspace
5665      \bbl@exp{\bbl@intraspace\bbl@cl{intsp}\bbl@@}%
5666      \directlua{
5667        Babel.sea_ranges = Babel.sea_ranges or {}
5668        Babel.set_chranges('\bbl@cl{sbcpr}',
5669                          '\bbl@cl{chrng}')
5670      }%
5671      \ifx\bbl@KVP@intrapenalty\@nnil
5672        \bbl@intrapenalty0\@@
5673      \fi
5674    \fi
5675  \fi
5676  \ifx\bbl@KVP@intrapenalty\@nnil\else
5677    \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5678  \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.

```

5679 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5680 \def\bblar@chars{%
5681   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5682   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5683   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5684 \def\bblar@elongated{%
5685   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5686   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5687   0649,064A}
5688 \begingroup
5689   \catcode`_ =11 \catcode`\:=11
5690   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5691 \endgroup
5692 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5693   \let\bbl@arabicjust\relax
5694   \newattribute\bblar@kashida
5695   \directlua{ Babel.attr_kashida = luatexbase.registernumber'\bblar@kashida' }%
5696   \bblar@kashida=\z@
5697   \bbl@patchfont{\bbl@parsejalt}}%
5698   \directlua{
5699     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5700     Babel.arabic.elong_map[\the\localeid] = {}
5701     luatexbase.add_to_callback('post_linebreak_filter',
5702                               Babel.arabic.justify, 'Babel.arabic.justify')
5703     luatexbase.add_to_callback('hpack_filter',
5704                               Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5705   }}%

```

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

```

5706 \def\bblar@fetchjalt#1#2#3#4{%
5707   \bbl@exp{\bbl@foreach{#1}}{%
5708     \bbl@ifunset{\bblar@JE@##1}%
5709     {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5710     {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse{\bblar@JE@##1}#2}}%
5711   \directlua{%
5712     local last = nil
5713     for item in node.traverse(tex.box[0].head) do
5714       if item.id == node.id'glyph' and item.char > 0x600 and

```

```

5715         not (item.char == 0x200D) then
5716             last = item
5717         end
5718     end
5719     Babel.arabic.#3['##1#4'] = last.char
5720 }}}}

```

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

```

5721 \gdef\bbl@parsejalt{%
5722 \ifx\addfontfeature\undefined\else
5723 \bbl@xin{/e}{/\bbl@cl{\lnbrk}}}%
5724 \fin@
5725 \directlua{%
5726     if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5727         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5728         tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5729     end
5730 }%
5731 \fi
5732 \fi}
5733 \gdef\bbl@parsejalti{%
5734 \begingroup
5735 \let\bbl@parsejalt\relax % To avoid infinite loop
5736 \edef\bbl@tempb{\fontid\font}%
5737 \bblar@nofswarn
5738 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5739 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5740 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5741 \addfontfeature{RawFeature+=jalt}%
5742 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5743 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5744 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5745 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5746 \directlua{%
5747     for k, v in pairs(Babel.arabic.from) do
5748         if Babel.arabic.dest[k] and
5749             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5750             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5751             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5752         end
5753     end
5754 }%
5755 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5756 \begingroup
5757 \catcode`#=11
5758 \catcode`~=11
5759 \directlua{
5760
5761 Babel.arabic = Babel.arabic or {}
5762 Babel.arabic.from = {}
5763 Babel.arabic.dest = {}
5764 Babel.arabic.justify_factor = 0.95
5765 Babel.arabic.justify_enabled = true
5766 Babel.arabic.kashida_limit = -1
5767
5768 function Babel.arabic.justify(head)
5769     if not Babel.arabic.justify_enabled then return head end
5770     for line in node.traverse_id(node.id'hlist', head) do
5771         Babel.arabic.justify_hlist(head, line)
5772     end
5773     return head

```

```

5774 end
5775
5776 function Babel.arabic.justify_hbox(head, gc, size, pack)
5777   local has_inf = false
5778   if Babel.arabic.justify_enabled and pack == 'exactly' then
5779     for n in node.traverse_id(12, head) do
5780       if n.stretch_order > 0 then has_inf = true end
5781     end
5782     if not has_inf then
5783       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5784     end
5785   end
5786   return head
5787 end
5788
5789 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5790   local d, new
5791   local k_list, k_item, pos_inline
5792   local width, width_new, full, k_curr, wt_pos, goal, shift
5793   local subst_done = false
5794   local elong_map = Babel.arabic.elong_map
5795   local cnt
5796   local last_line
5797   local GLYPH = node.id'glyph'
5798   local KASHIDA = Babel.attr_kashida
5799   local LOCALE = Babel.attr_locale
5800
5801   if line == nil then
5802     line = {}
5803     line.glue_sign = 1
5804     line.glue_order = 0
5805     line.head = head
5806     line.shift = 0
5807     line.width = size
5808   end
5809
5810   % Exclude last line. todo. But-- it discards one-word lines, too!
5811   % ? Look for glue = 12:15
5812   if (line.glue_sign == 1 and line.glue_order == 0) then
5813     elongs = {} % Stores elongated candidates of each line
5814     k_list = {} % And all letters with kashida
5815     pos_inline = 0 % Not yet used
5816
5817     for n in node.traverse_id(GLYPH, line.head) do
5818       pos_inline = pos_inline + 1 % To find where it is. Not used.
5819
5820       % Elongated glyphs
5821       if elong_map then
5822         local locale = node.get_attribute(n, LOCALE)
5823         if elong_map[locale] and elong_map[locale][n.font] and
5824           elong_map[locale][n.font][n.char] then
5825           table.insert(elongs, {node = n, locale = locale} )
5826           node.set_attribute(n.prev, KASHIDA, 0)
5827         end
5828       end
5829
5830       % Tatwil
5831       if Babel.kashida_wts then
5832         local k_wt = node.get_attribute(n, KASHIDA)
5833         if k_wt > 0 then % todo. parameter for multi inserts
5834           table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5835         end
5836       end
5837     end

```

```

5837
5838     end % of node.traverse_id
5839
5840     if #elongs == 0 and #k_list == 0 then goto next_line end
5841     full = line.width
5842     shift = line.shift
5843     goal = full * Babel.arabic.justify_factor % A bit crude
5844     width = node.dimensions(line.head) % The 'natural' width
5845
5846     % == Elongated ==
5847     % Original idea taken from 'chickenize'
5848     while (#elongs > 0 and width < goal) do
5849         subst_done = true
5850         local x = #elongs
5851         local curr = elongs[x].node
5852         local oldchar = curr.char
5853         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5854         width = node.dimensions(line.head) % Check if the line is too wide
5855         % Substitute back if the line would be too wide and break:
5856         if width > goal then
5857             curr.char = oldchar
5858             break
5859         end
5860         % If continue, pop the just substituted node from the list:
5861         table.remove(elongs, x)
5862     end
5863
5864     % == Tatwil ==
5865     if #k_list == 0 then goto next_line end
5866
5867     width = node.dimensions(line.head) % The 'natural' width
5868     k_curr = #k_list % Traverse backwards, from the end
5869     wt_pos = 1
5870
5871     while width < goal do
5872         subst_done = true
5873         k_item = k_list[k_curr].node
5874         if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5875             d = node.copy(k_item)
5876             d.char = 0x0640
5877             d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5878             d.xoffset = 0
5879             line.head, new = node.insert_after(line.head, k_item, d)
5880             width_new = node.dimensions(line.head)
5881             if width > goal or width == width_new then
5882                 node.remove(line.head, new) % Better compute before
5883                 break
5884             end
5885             if Babel.fix_diacr then
5886                 Babel.fix_diacr(k_item.next)
5887             end
5888             width = width_new
5889         end
5890         if k_curr == 1 then
5891             k_curr = #k_list
5892             wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5893         else
5894             k_curr = k_curr - 1
5895         end
5896     end
5897
5898     % Limit the number of tatweel by removing them. Not very efficient,
5899     % but it does the job in a quite predictable way.

```

```

5900   if Babel.arabic.kashida_limit > -1 then
5901       cnt = 0
5902       for n in node.traverse_id(GLYPH, line.head) do
5903           if n.char == 0x0640 then
5904               cnt = cnt + 1
5905               if cnt > Babel.arabic.kashida_limit then
5906                   node.remove(line.head, n)
5907               end
5908           else
5909               cnt = 0
5910           end
5911       end
5912   end
5913
5914   ::next_line::
5915
5916   % Must take into account marks and ins, see luatex manual.
5917   % Have to be executed only if there are changes. Investigate
5918   % what's going on exactly.
5919   if subst_done and not gc then
5920       d = node.hpack(line.head, full, 'exactly')
5921       d.shift = shift
5922       node.insert_before(head, line, d)
5923       node.remove(head, line)
5924   end
5925   end % if process line
5926 end
5927 }
5928 \endgroup
5929 \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`.

```

5930 \def\bbl@scr@node@list{%
5931   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
5932   ,Greek,Latin,Old Church Slavonic Cyrillic,}
5933 \ifnum\bbl@bidimode=102 % bidi-r
5934   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
5935 \fi
5936 \def\bbl@set@renderer{%
5937   \bbl@xin@{\bbl@c\sname}}{\bbl@scr@node@list}%
5938   \ifin@
5939     \let\bbl@unset@renderer\relax
5940   \else
5941     \bbl@exp{%
5942       \def\\bbl@unset@renderer{%
5943         \def<g__fontspec_default_fontopts_clist>{%
5944           \[g__fontspec_default_fontopts_clist]}}%
5945       \def<g__fontspec_default_fontopts_clist>{%
5946         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
5947     \fi}
5948 <@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.

```

5949 % TODO - to a lua file
5950 \directlua{% DL6
5951 Babel.script_blocks = {
5952   ['dflt'] = {},
5953   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5954               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5955   ['Armn'] = {{0x0530, 0x058F}},
5956   ['Beng'] = {{0x0980, 0x09FF}},
5957   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5958   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5959   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5960              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5961   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5962   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5963              {0xAB00, 0xAB2F}},
5964   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5965   % Don't follow strictly Unicode, which places some Coptic letters in
5966   % the 'Greek and Coptic' block
5967   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5968   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5969              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5970              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5971              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5972              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5973              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5974   ['Hebr'] = {{0x0590, 0x05FF}},
5975   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5976              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5977   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5978   ['Knda'] = {{0x0C80, 0x0CFF}},
5979   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5980              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5981              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5982   ['Lao'] = {{0x0E80, 0x0EFF}},
5983   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5984              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5985              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5986   ['Mahj'] = {{0x11150, 0x1117F}},
5987   ['Mlym'] = {{0x0D00, 0x0D7F}},
5988   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5989   ['Orya'] = {{0x0B00, 0x0B7F}},
5990   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5991   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5992   ['Taml'] = {{0x0B80, 0x0BFF}},
5993   ['Telu'] = {{0x0C00, 0x0C7F}},
5994   ['Tfng'] = {{0x2D30, 0x2D7F}},
5995   ['Thai'] = {{0x0E00, 0x0E7F}},
5996   ['Tibt'] = {{0x0F00, 0x0FFF}},
5997   ['Vaii'] = {{0xA500, 0xA63F}},
5998   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5999 }
6000
6001 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6002 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6003 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6004
6005 function Babel.locale_map(head)
6006   if not Babel.locale_mapped then return head end

```

```

6007
6008 local LOCALE = Babel.attr_locale
6009 local GLYPH = node.id('glyph')
6010 local inmath = false
6011 local toloc_save
6012 for item in node.traverse(head) do
6013   local toloc
6014   if not inmath and item.id == GLYPH then
6015     % Optimization: build a table with the chars found
6016     if Babel.chr_to_loc[item.char] then
6017       toloc = Babel.chr_to_loc[item.char]
6018     else
6019       for lc, maps in pairs(Babel.loc_to_scr) do
6020         for _, rg in pairs(maps) do
6021           if item.char >= rg[1] and item.char <= rg[2] then
6022             Babel.chr_to_loc[item.char] = lc
6023             toloc = lc
6024             break
6025           end
6026         end
6027       end
6028       % Treat composite chars in a different fashion, because they
6029       % 'inherit' the previous locale.
6030       if (item.char >= 0x0300 and item.char <= 0x036F) or
6031          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6032          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6033         Babel.chr_to_loc[item.char] = -2000
6034         toloc = -2000
6035       end
6036       if not toloc then
6037         Babel.chr_to_loc[item.char] = -1000
6038       end
6039     end
6040     if toloc == -2000 then
6041       toloc = toloc_save
6042     elseif toloc == -1000 then
6043       toloc = nil
6044     end
6045     if toloc and Babel.locale_props[toloc] and
6046        Babel.locale_props[toloc].letters and
6047        tex.getcatcode(item.char) \string~= 11 then
6048       toloc = nil
6049     end
6050     if toloc and Babel.locale_props[toloc].script
6051        and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6052        and Babel.locale_props[toloc].script ==
6053        Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6054       toloc = nil
6055     end
6056     if toloc then
6057       if Babel.locale_props[toloc].lg then
6058         item.lang = Babel.locale_props[toloc].lg
6059         node.set_attribute(item, LOCALE, toloc)
6060       end
6061       if Babel.locale_props[toloc]['/'..item.font] then
6062         item.font = Babel.locale_props[toloc]['/'..item.font]
6063       end
6064     end
6065     toloc_save = toloc
6066   elseif not inmath and item.id == 7 then % Apply recursively
6067     item.replace = item.replace and Babel.locale_map(item.replace)
6068     item.pre = item.pre and Babel.locale_map(item.pre)
6069     item.post = item.post and Babel.locale_map(item.post)

```

```

6070 elseif item.id == node.id'math' then
6071   inmath = (item.subtype == 0)
6072 end
6073 end
6074 return head
6075 end
6076 }

```

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

```

6077 \newcommand\babelcharproperty[1]{%
6078   \count@=#1\relax
6079   \ifvmode
6080     \expandafter\babel@chprop
6081   \else
6082     \babel@error{charproperty-only-vertical}{}{}{}%
6083   \fi}
6084 \newcommand\babel@chprop[3][\the\count@]{%
6085   \@tempcnta=#1\relax
6086   \babel@ifunset{babel@chprop@#2}% {unknown-char-property}
6087   {\babel@error{unknown-char-property}{}{#2}{}%
6088   }%
6089   \loop
6090     \babel@cs{chprop@#2}{#3}%
6091   \ifnum\count@<\@tempcnta
6092     \advance\count@\@ne
6093   \repeat}
6094 \def\babel@chprop@direction#1{%
6095   \directlua{
6096     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6097     Babel.characters[\the\count@]['d'] = '#1'
6098   }}
6099 \let\babel@chprop@bc\babel@chprop@direction
6100 \def\babel@chprop@mirror#1{%
6101   \directlua{
6102     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6103     Babel.characters[\the\count@]['m'] = '\number#1'
6104   }}
6105 \let\babel@chprop@bmg\babel@chprop@mirror
6106 \def\babel@chprop@linebreak#1{%
6107   \directlua{
6108     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6109     Babel.cjk_characters[\the\count@]['c'] = '#1'
6110   }}
6111 \let\babel@chprop@lb\babel@chprop@linebreak
6112 \def\babel@chprop@locale#1{%
6113   \directlua{
6114     Babel.chr_to_loc = Babel.chr_to_loc or {}
6115     Babel.chr_to_loc[\the\count@] =
6116       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
6117   }}

```

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.

```

6118 \directlua{% DL7
6119   Babel.nohyphenation = \the\l@nohyphenation
6120 }

```

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 in 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).

```

6121 \begingroup
6122 \catcode`\~ = 12
6123 \catcode`\% = 12
6124 \catcode`\& = 14
6125 \catcode`\| = 12
6126 \gdef\babelprehyphenation{%&
6127   \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6128 \gdef\babelposthyphenation{%&
6129   \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6130 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6131   \ifcase#1
6132     \bbl@activateprehyphen
6133   \or
6134     \bbl@activateposthyphen
6135   \fi
6136 \begingroup
6137   \def\babeltempa{\bbl@add@list\babeltempb}%&
6138   \let\babeltempb\empty
6139   \def\bbl@tempa{#5}%&
6140   \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6141   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6142     \bbl@ifsamestring{##1}{remove}%&
6143     {\bbl@add@list\babeltempb{nil}}}%&
6144     {\directlua{
6145       local rep = [=##1]=]
6146       local three_args = '%s*=%s*([%-d%.%a{}|])+)%s+([%-d%.%a{}|])+)%s+([%-d%.%a{}|])+)'
6147       & Numeric passes directly: kern, penalty...
6148       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6149       rep = rep:gsub('^%s*(insert)%s*', ', 'insert = true, ')
6150       rep = rep:gsub('^%s*(after)%s*', ', 'after = true, ')
6151       rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6152       rep = rep:gsub('node%s*=%s*([a+)%s*([a+)%s*([a+)%s*', Babel.capture_node)
6153       rep = rep:gsub(' (norule)' .. three_args,
6154         'norule = {' .. '%2, %3, %4' .. '}')
6155       if #1 == 0 or #1 == 2 then
6156         rep = rep:gsub(' (space)' .. three_args,
6157           'space = {' .. '%2, %3, %4' .. '}')
6158         rep = rep:gsub(' (spacefactor)' .. three_args,
6159           'spacefactor = {' .. '%2, %3, %4' .. '}')
6160         rep = rep:gsub(' (kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6161         & Transform values
6162         rep, n = rep:gsub(' {([%a%-%.]+)|([%a%_%.]+)}',
6163           function(v,d)
6164             return string.format (
6165               '{\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6166               v,
6167               load( 'return Babel.locale_props'..
6168                 '{\the\csname bbl@id@@#3\endcsname}.' .. d)() )
6169             end )
6170         rep, n = rep:gsub(' {([%a%-%.]+)|([%-d%.]+)}',
6171           '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6172       end
6173       if #1 == 1 then
6174         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6175         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6176         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6177       end
6178       tex.print([[string\babeltempa{[]] .. rep .. [{}]])
6179     }]}%&
6180   \bbl@foreach\babeltempb{%&

```

```

6181 \bbl@forkv{##1}{&%
6182 \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6183 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6184 \ifin@else
6185 \bbl@error{bad-transform-option}{###1}{&%
6186 \fi}}&%
6187 \let\bbl@kv@attribute\relax
6188 \let\bbl@kv@label\relax
6189 \let\bbl@kv@fonts\empty
6190 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6191 \ifx\bbl@kv@fonts\empty\else\bbl@settransfont\fi
6192 \ifx\bbl@kv@attribute\relax
6193 \ifx\bbl@kv@label\relax\else
6194 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6195 \bbl@replace\bbl@kv@fonts{ },}&%
6196 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3\bbl@kv@fonts}&%
6197 \count@ \z@
6198 \def\bbl@elt##1##2##3{&%
6199 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6200 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6201 {\count@\@ne}&%
6202 {\bbl@error{font-conflict-transforms}{}}}&%
6203 {}}&%
6204 \bbl@transfont@list
6205 \ifnum\count@=\z@
6206 \bbl@exp{\global\bbl@add\bbl@transfont@list
6207 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6208 \fi
6209 \bbl@ifunset{\bbl@kv@attribute}&%
6210 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6211 {}}&%
6212 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6213 \fi
6214 \else
6215 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6216 \fi
6217 \directlua{
6218 local lbr = Babel.linebreaking.replacements[#1]
6219 local u = unicode.utf8
6220 local id, attr, label
6221 if #1 == 0 then
6222 id = \the\csname bbl@id@@#3\endcsname\space
6223 else
6224 id = \the\csname l@#3\endcsname\space
6225 end
6226 \ifx\bbl@kv@attribute\relax
6227 attr = -1
6228 \else
6229 attr = luatexbase.registernumber'\bbl@kv@attribute'
6230 \fi
6231 \ifx\bbl@kv@label\relax\else &% Same refs:
6232 label = [==[\bbl@kv@label]==]
6233 \fi
6234 &% Convert pattern:
6235 local patt = string.gsub([==[#4]==], '%s', '')
6236 if #1 == 0 then
6237 patt = string.gsub(patt, '|', ' ')
6238 end
6239 if not u.find(patt, '()', nil, true) then
6240 patt = '()' .. patt .. '()'
6241 end
6242 if #1 == 1 then
6243 patt = string.gsub(patt, '%(%)%^', '^()')

```

```

6244     patt = string.gsub(patt, '%$%(%)', '()$')
6245     end
6246     patt = u.gsub(patt, '{(.)}',
6247         function (n)
6248             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6249         end)
6250     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6251         function (n)
6252             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6253         end)
6254     lbkr[id] = lbkr[id] or {}
6255     table.insert(lbkr[id],
6256         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6257 }&%
6258 \endgroup}
6259 \endgroup
6260 \let\bbl@transfont@list\empty
6261 \def\bbl@settransfont{%
6262     \global\let\bbl@settransfont\relax % Execute only once
6263     \gdef\bbl@transfont{%
6264         \def\bbl@elt####1####2####3{%
6265             \bbl@ifblank{####3}%
6266             {\count@tw@}% Do nothing if no fonts
6267             {\count@z@
6268                 \bbl@vforeach{####3}{%
6269                     \def\bbl@tempd{#####1}%
6270                     \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6271                     \ifx\bbl@tempd\bbl@tempe
6272                         \count@@ne
6273                     \else\ifx\bbl@tempd\bbl@transfam
6274                         \count@@ne
6275                     \fi\fi}%
6276                 \ifcase\count@
6277                     \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6278                 \or
6279                     \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6280                 \fi}}%
6281         \bbl@transfont@list}%
6282     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6283     \gdef\bbl@transfam{-unknown-}%
6284     \bbl@foreach\bbl@font@fams{%
6285         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6286         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6287         {\xdef\bbl@transfam{##1}}%
6288     }}}}
6289 \DeclareRobustCommand\enablelocaletransform[1]{%
6290     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6291     {\bbl@error{transform-not-available}{#1}{}}}%
6292     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6293 \DeclareRobustCommand\disablelocaletransform[1]{%
6294     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6295     {\bbl@error{transform-not-available-b}{#1}{}}}%
6296     {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}%
6297 \def\bbl@activateposthyphen{%
6298     \let\bbl@activateposthyphen\relax
6299     \ifx\bbl@attr@hboxed\undefined
6300         \newattribute\bbl@attr@hboxed
6301     \fi
6302     \directlua{
6303         require('babel-transforms.lua')
6304         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6305     }}
6306 \def\bbl@activateprehyphen{%

```

```

6307 \let\bbl@activateprehyphen\relax
6308 \ifx\bbl@attr@hboxed\undefined
6309 \newattribute\bbl@attr@hboxed
6310 \fi
6311 \directlua{
6312   require('babel-transforms.lua')
6313   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6314 }
6315 \newcommand\SetTransformValue[3]{%
6316 \directlua{
6317   Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6318 }

```

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.

```

6319 \newcommand\localeprehyphenation[1]{%
6320 \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.

```

6321 \def\bbl@activate@preotf{%
6322 \let\bbl@activate@preotf\relax % only once
6323 \directlua{
6324   function Babel.pre_otfload_v(head)
6325     if Babel.numbers and Babel.digits_mapped then
6326       head = Babel.numbers(head)
6327     end
6328     if Babel.bidi_enabled then
6329       head = Babel.bidi(head, false, dir)
6330     end
6331     return head
6332   end
6333   %
6334   function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6335     if Babel.numbers and Babel.digits_mapped then
6336       head = Babel.numbers(head)
6337     end
6338     if Babel.bidi_enabled then
6339       head = Babel.bidi(head, false, dir)
6340     end
6341     return head
6342   end
6343   %
6344   luatexbase.add_to_callback('pre_linebreak_filter',
6345     Babel.pre_otfload_v,
6346     'Babel.pre_otfload_v',
6347     luatexbase.priority_in_callback('pre_linebreak_filter',
6348       'luaotfload.node_processor') or nil)
6349   %
6350   luatexbase.add_to_callback('hpack_filter',
6351     Babel.pre_otfload_h,
6352     'Babel.pre_otfload_h',
6353     luatexbase.priority_in_callback('hpack_filter',
6354       'luaotfload.node_processor') or nil)
6355 }

```

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.

```

6356 \breakafterdirmode=1
6357 \ifnum\bbbl@bidimode>\@ne % Any bidi= except default (=1)
6358   \let\bbbl@beforeforeign\leavevmode
6359   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6360   \RequirePackage{luatexbase}
6361   \bbbl@activate@preotf
6362   \directlua{
6363     require('babel-data-bidi.lua')
6364     \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
6365       require('babel-bidi-basic.lua')
6366     \or
6367       require('babel-bidi-basic-r.lua')
6368       table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6369       table.insert(Babel.ranges, {0xF0000, 0xFFFFD, 'on'})
6370       table.insert(Babel.ranges, {0x100000, 0x10FFFD, 'on'})
6371     \fi}
6372   \newattribute\bbbl@attr@dir
6373   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
6374   \bbbl@exp{\output{\bodydir\pagedir\the\output}}
6375 \fi
6376 \chardef\bbbl@thetextdir\z@
6377 \chardef\bbbl@thepardir\z@
6378 \def\bbbl@getluadir#1{%
6379   \directlua{
6380     if tex.#ldir == 'TLT' then
6381       tex.sprint('0')
6382     elseif tex.#ldir == 'TRT' then
6383       tex.sprint('1')
6384     end}}
6385 \def\bbbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/l rl
6386   \ifcase#3\relax
6387     \ifcase\bbbl@getluadir{#1}\relax\else
6388       #2 TLT\relax
6389     \fi
6390   \else
6391     \ifcase\bbbl@getluadir{#1}\relax
6392       #2 TRT\relax
6393     \fi
6394   \fi}
6395 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6396 \def\bbbl@thedir{0}
6397 \def\bbbl@textdir#1{%
6398   \bbbl@setluadir{text}\textdir{#1}%
6399   \chardef\bbbl@thetextdir#1\relax
6400   \edef\bbbl@thedir{\the\numexpr\bbbl@thepardir*4+#1}%
6401   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*4+#1}}
6402 \def\bbbl@pardir#1{% Used twice
6403   \bbbl@setluadir{par}\pardir{#1}%
6404   \chardef\bbbl@thepardir#1\relax}
6405 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}% Used once
6406 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}% Unused
6407 \def\bbbl@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.

```

6408 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6409   \def\bbbl@insidemath{0}%
6410   \def\bbbl@everymath{\def\bbbl@insidemath{1}}
6411   \def\bbbl@everydisplay{\def\bbbl@insidemath{2}}
6412   \frozen@everymath\expandafter{%
6413     \expandafter\bbbl@everymath\the\frozen@everymath}

```

```

6414 \frozen@everydisplay\expandafter{%
6415   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6416 \AtBeginDocument{
6417   \directlua{
6418     function Babel.math_box_dir(head)
6419       if not (token.get_macro('bbl@insidemath') == '0') then
6420         if Babel.hlist_has_bidi(head) then
6421           local d = node.new(node.id'dir')
6422           d.dir = '+TRT'
6423           node.insert_before(head, node.has_glyph(head), d)
6424           local inmath = false
6425           for item in node.traverse(head) do
6426             if item.id == 11 then
6427               inmath = (item.subtype == 0)
6428             elseif not inmath then
6429               node.set_attribute(item,
6430                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6431             end
6432           end
6433         end
6434       end
6435       return head
6436     end
6437     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6438       "Babel.math_box_dir", 0)
6439     if Babel.unset_atdir then
6440       luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6441         "Babel.unset_atdir")
6442       luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6443         "Babel.unset_atdir")
6444     end
6445   } }%
6446 \fi

Experimental. Tentative name.

6447 \DeclareRobustCommand\localebox[1]{%
6448   {\def\bbl@insidemath{0}%
6449     \mbox{\foreignlanguage{\language}{#1}}}}

```

10.12 Layout

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

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

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

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

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

```

6450 \bbl@trace{Redefinitions for bidi layout}
6451 %
6452 <<(*More package options)>> ≡

```

```

6453 \chardef\bbl@eqnpos\z@
6454 \DeclareOption{leqno}{\chardef\bbl@eqnpos@ne}
6455 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6456 <</More package options>>
6457 %
6458 \ifnum\bbl@bidimode>\z@ % Any bidi=
6459 \matheqdirmode@ne % A luatex primitive
6460 \let\bbl@eqnodir\relax
6461 \def\bbl@eqdel{()}
6462 \def\bbl@eqnum{%
6463   {\normalfont\normalcolor
6464     \expandafter\@firstoftwo\bbl@eqdel
6465     \theequation
6466     \expandafter\@secondoftwo\bbl@eqdel}}
6467 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6468 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6469 \def\bbl@eqno@flip#1{%
6470   \ifdim\predisplaysize=-\maxdimen
6471     \eqno
6472     \hb@xt@.01pt{%
6473       \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6474   \else
6475     \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6476   \fi
6477   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6478 \def\bbl@leqno@flip#1{%
6479   \ifdim\predisplaysize=-\maxdimen
6480     \leqno
6481     \hb@xt@.01pt{%
6482       \hss\hb@xt@\displaywidth{#1}\glet\bbl@upset\@currentlabel}\hss}%
6483   \else
6484     \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6485   \fi
6486   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6487 \AtBeginDocument{%
6488   \ifx\bbl@noamsmath\relax\else
6489     \ifx\maketag@@@undefined % Normal equation, eqnarray
6490       \AddToHook{env/equation/begin}{%
6491         \ifnum\bbl@thetextdir>\z@
6492           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6493           \let\@eqnnum\bbl@eqnum
6494           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6495           \chardef\bbl@thetextdir\z@
6496           \bbl@add\normalfont{\bbl@eqnodir}%
6497           \ifcase\bbl@eqnpos
6498             \let\bbl@puteqno\bbl@eqno@flip
6499           \or
6500             \let\bbl@puteqno\bbl@leqno@flip
6501           \fi
6502         \fi}%
6503       \ifnum\bbl@eqnpos=\tw@\else
6504         \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6505       \fi
6506       \AddToHook{env/eqnarray/begin}{%
6507         \ifnum\bbl@thetextdir>\z@
6508           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6509           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6510           \chardef\bbl@thetextdir\z@
6511           \bbl@add\normalfont{\bbl@eqnodir}%
6512         \ifnum\bbl@eqnpos=@ne
6513           \def\@eqnnum{%
6514             \setbox\z@\hbox{\bbl@eqnum}%
6515             \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%

```

```

6516         \else
6517         \let\@eqnnum\bbledqnum
6518         \fi
6519     \fi}
6520 % Hack. YA luatex bug?:
6521 \expandafter\bbledsreplace\csname] \endcsname{$$}{\eqno\kern.001pt$}$}%
6522 \else % amstex
6523     \bbledexp{% Hack to hide maybe undefined conditionals:
6524         \chardef\bbledqnpos=0%
6525         \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6526     \ifnum\bbledqnpos=\@ne
6527         \let\bbledams@lap\hbox
6528     \else
6529         \let\bbledams@lap\llap
6530     \fi
6531 \ExplSyntaxOn % Required by \bbledsreplace with \intertext@
6532 \bbledsreplace\intertext@{\normalbaselines}%
6533     {\normalbaselines
6534     \ifx\bbledeqnodir\relax\else\bbledpdir\@ne\bbledeqnodir\fi}%
6535 \ExplSyntaxOff
6536 \def\bbledams@tagbox#1#2{#1{\bbledeqnodir#2}}% #1=hbox|@lap|flip
6537 \ifx\bbledams@lap\hbox % leqno
6538     \def\bbledams@flip#1{%
6539         \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}}%
6540 \else % eqno
6541     \def\bbledams@flip#1{%
6542         \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}}%
6543 \fi
6544 \def\bbledams@preset#1{%
6545     \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6546     \ifnum\bbledthetextdir>\z@
6547         \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6548         \bbledsreplace\textdef@{\hbox}{\bbledams@tagbox\hbox}%
6549         \bbledsreplace\maketag@@@{\hbox}{\bbledams@tagbox#1}%
6550     \fi}%
6551 \ifnum\bbledqnpos=\tw@ \else
6552     \def\bbledams@equation{%
6553         \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6554         \ifnum\bbledthetextdir>\z@
6555             \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6556             \chardef\bbledthetextdir\z@
6557             \bbledadd\normalfont{\bbledeqnodir}%
6558             \ifcase\bbledqnpos
6559                 \def\veqno##1##2{\bbledeqno@flip{##1##2}}%
6560             \or
6561                 \def\veqno##1##2{\bbledleqno@flip{##1##2}}%
6562             \fi
6563         \fi}%
6564     \AddToHook{env/equation/begin}{\bbledams@equation}%
6565     \AddToHook{env/equation*/begin}{\bbledams@equation}%
6566 \fi
6567 \AddToHook{env/cases/begin}{\bbledams@preset\bbledams@lap}%
6568 \AddToHook{env/multline/begin}{\bbledams@preset\hbox}%
6569 \AddToHook{env/gather/begin}{\bbledams@preset\bbledams@lap}%
6570 \AddToHook{env/gather*/begin}{\bbledams@preset\bbledams@lap}%
6571 \AddToHook{env/align/begin}{\bbledams@preset\bbledams@lap}%
6572 \AddToHook{env/align*/begin}{\bbledams@preset\bbledams@lap}%
6573 \AddToHook{env/alignat/begin}{\bbledams@preset\bbledams@lap}%
6574 \AddToHook{env/alignat*/begin}{\bbledams@preset\bbledams@lap}%
6575 \AddToHook{env/eqnalign/begin}{\bbledams@preset\hbox}%
6576 % Hackish, for proper alignment. Don't ask me why it works!:
6577 \bbledexp{% Avoid a 'visible' conditional
6578     \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%

```



```

6579     \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6580 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6581 \AddToHook{env/split/before}{%
6582   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6583   \ifnum\bbl@thetextdir>\z@
6584     \bbl@ifsamestring\@currentvir{equation}%
6585     {\ifx\bbl@ams@lap\hbox % leqno
6586       \def\bbl@ams@flip#1{%
6587         \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6588       \else
6589         \def\bbl@ams@flip#1{%
6590           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#{1}}}}%
6591         \fi}%
6592     }%
6593   \fi}%
6594 \fi\fi}
6595 \fi
6596 \def\bbl@provide@extra#1{%
6597   % == onchar ==
6598   \ifx\bbl@KVP@onchar\@nnil\else
6599     \bbl@luahyphenate
6600     \bbl@exp{%
6601       \\\AddToHook{env/document/before}{{\select@language{#1}}}}%
6602     \directlua{
6603       if Babel.locale_mapped == nil then
6604         Babel.locale_mapped = true
6605         Babel.linebreaking.add_before(Babel.locale_map, 1)
6606         Babel.loc_to_scr = {}
6607         Babel.chr_to_loc = Babel.chr_to_loc or {}
6608       end
6609       Babel.locale_props[\the\localeid].letters = false
6610     }%
6611     \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6612     \ifin@
6613       \directlua{
6614         Babel.locale_props[\the\localeid].letters = true
6615       }%
6616     \fi
6617     \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6618     \ifin@
6619       \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6620         \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6621       \fi
6622       \bbl@exp{\bbl@add\bbl@starthyphens
6623         {\bbl@patterns@lua{\language\language}}}%
6624       %^A add error/warning if no script
6625       \directlua{
6626         if Babel.script_blocks['\bbl@cl{sbc}'] then
6627           Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6628           Babel.locale_props[\the\localeid].lg = \the\@nameuse{l\language}\space
6629         end
6630       }%
6631     \fi
6632     \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6633     \ifin@
6634       \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}}}%
6635       \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}}}%
6636     \directlua{
6637       if Babel.script_blocks['\bbl@cl{sbc}'] then
6638         Babel.loc_to_scr[\the\localeid] =
6639         Babel.script_blocks['\bbl@cl{sbc}']
6640       end}%
6641     \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont

```

```

6642 \AtBeginDocument{%
6643   \bbl@patchfont{\bbl@mapselect}}%
6644   {\selectfont}}%
6645 \def\bbl@mapselect{%
6646   \let\bbl@mapselect\relax
6647   \edef\bbl@prefontid{\fontid\font}}%
6648 \def\bbl@mapdir##1{%
6649   \begingroup
6650     \setbox\z@\hbox{% Force text mode
6651       \def\language{##1}%
6652       \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6653       \bbl@switchfont
6654       \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6655         \directlua{
6656           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6657             ['\bbl@prefontid'] = \fontid\font\space}%
6658         \fi}%
6659     \endgroup}%
6660 \fi
6661 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir\language}}}%
6662 \fi
6663 % TODO - catch non-valid values
6664 \fi
6665 % == mapfont ==
6666 % For bidi texts, to switch the font based on direction
6667 \ifx\bbl@KVP@mapfont\@nnil\else
6668   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{%
6669     {\bbl@error{unknown-mapfont}}}%
6670   \bbl@ifunset{\bbl@lsys\language}{\bbl@provide@lsys\language}}{%
6671     \bbl@ifunset{\bbl@wdir\language}{\bbl@provide@dirs\language}}{%
6672     \ifx\bbl@mapselect\@undefined % TODO. See onchar.
6673       \AtBeginDocument{%
6674         \bbl@patchfont{\bbl@mapselect}}%
6675         {\selectfont}}%
6676       \def\bbl@mapselect{%
6677         \let\bbl@mapselect\relax
6678         \edef\bbl@prefontid{\fontid\font}}%
6679       \def\bbl@mapdir##1{%
6680         {\def\language{##1}%
6681           \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6682           \bbl@switchfont
6683           \directlua{Babel.fontmap
6684             [\the\csname bbl@wdir##1\endcsname]%
6685             [\bbl@prefontid]=\fontid\font}}}%
6686       \fi
6687       \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir\language}}}%
6688     \fi
6689 % == Line breaking: CJK quotes == %^A -> @extras
6690 \ifcase\bbl@engine\or
6691   \bbl@xin{/c}{\bbl@cl{\lnbrk}}%
6692   \ifin@
6693     \bbl@ifunset{\bbl@quote\language}{%
6694       {\directlua{
6695         Babel.locale_props[\the\localeid].cjk_quotes = {}
6696         local cs = 'op'
6697         for c in string.utfvalues(
6698           [[\csname bbl@quote\language\endcsname]]) do
6699           if Babel.cjk_characters[c].c == 'qu' then
6700             Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6701             end
6702             cs = ( cs == 'op') and 'cl' or 'op'
6703           end
6704         }}%

```

```

6705 \fi
6706 \fi
6707 % == Counters: mapdigits ==
6708 % Native digits
6709 \ifx\bbbl@KVP@mapdigits\@nnil\else
6710 \bbl@ifunset{bbl@dgnat@\language\name}{}%
6711 {\RequirePackage{luatexbase}%
6712 \bbl@activate@preotf
6713 \directlua{
6714 Babel.digits_mapped = true
6715 Babel.digits = Babel.digits or {}
6716 Babel.digits[\the\localeid] =
6717 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6718 if not Babel.numbers then
6719 function Babel.numbers(head)
6720 local LOCALE = Babel.attr_locale
6721 local GLYPH = node.id'glyph'
6722 local inmath = false
6723 for item in node.traverse(head) do
6724 if not inmath and item.id == GLYPH then
6725 local temp = node.get_attribute(item, LOCALE)
6726 if Babel.digits[temp] then
6727 local chr = item.char
6728 if chr > 47 and chr < 58 then
6729 item.char = Babel.digits[temp][chr-47]
6730 end
6731 end
6732 elseif item.id == node.id'math' then
6733 inmath = (item.subtype == 0)
6734 end
6735 end
6736 return head
6737 end
6738 end
6739 }}%
6740 \fi
6741 % == transforms ==
6742 \ifx\bbbl@KVP@transforms\@nnil\else
6743 \def\bbl@elt##1##2##3{%
6744 \in@{$transforms.}{##1}%
6745 \ifin@
6746 \def\bbl@tempa{##1}%
6747 \bbl@replace\bbl@tempa{transforms.}{}%
6748 \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6749 \fi}%
6750 \bbl@exp{%
6751 \\bbl@ifblank{\bbl@cl{dgnat}}}%
6752 {\let\\bbl@tempa\relax}%
6753 {\def\\bbl@tempa{%
6754 \\bbl@elt{transforms.prehyphenation}%
6755 {digits.native.1.0}{([0-9])}%
6756 \\bbl@elt{transforms.prehyphenation}%
6757 {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}%
6758 \ifx\bbl@tempa\relax\else
6759 \toks@{\expandafter\expandafter\expandafter{%
6760 \csname bbl@inidata@\language\name\endcsname}%
6761 \bbl@csarg\edef{inidata@\language\name}{%
6762 \unexpanded\expandafter{\bbl@tempa}%
6763 \the\toks@}%
6764 \fi
6765 \csname bbl@inidata@\language\name\endcsname
6766 \bbl@release@transforms\relax % \relax closes the last item.
6767 \fi}

```

Start tabular here:

```

6768 \def\localerestoredirs{%
6769   \ifcase\bbbl@thetextdir
6770     \ifnum\textdirection=\z@\else\textdir TLT\fi
6771   \else
6772     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6773   \fi
6774   \ifcase\bbbl@thepardir
6775     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6776   \else
6777     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6778   \fi}
6779 \IfBabelLayout{tabular}%
6780   {\chardef\bbbl@tabular@mode\tw@}% All RTL
6781   {\IfBabelLayout{notabular}%
6782     {\chardef\bbbl@tabular@mode\z@}%
6783     {\chardef\bbbl@tabular@mode\@ne}}% Mixed, with LTR cols
6784 \ifnum\bbbl@bidimode>\@ne % Any lua bidi= except default=1
6785 % Redefine: vrules mess up dirs. TODO: why?
6786 \def\@arstrut{\relax\copy\@arstrutbox}%
6787 \ifcase\bbbl@tabular@mode\or % 1 = Mixed - default
6788   \let\bbbl@parabefore\relax
6789   \AddToHook{para/before}{\bbbl@parabefore}
6790   \AtBeginDocument{%
6791     \bbbl@replace\@tabular{$}{$%
6792       \def\bbbl@insidemath{0}%
6793       \def\bbbl@parabefore{\localerestoredirs}}%
6794     \ifnum\bbbl@tabular@mode=\@ne
6795       \bbbl@ifunset{\@tabclassz}{}%
6796       \bbbl@exp{% Hide conditionals
6797         \\bbbl@sreplace\\ \@tabclassz
6798         {\<ifcase>\\ \@chnum}%
6799         {\localerestoredirs\<ifcase>\\ \@chnum}}}%
6800       \@ifpackageloaded{colortbl}%
6801       {\bbbl@sreplace\@classz
6802         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6803       {\@ifpackageloaded{array}%
6804         {\bbbl@exp{% Hide conditionals
6805           \\bbbl@sreplace\\ \@classz
6806           {\<ifcase>\\ \@chnum}%
6807           {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6808           \\bbbl@sreplace\\ \@classz
6809           {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6810         {}}}%
6811     \fi}%
6812 \or % 2 = All RTL - tabular
6813   \let\bbbl@parabefore\relax
6814   \AddToHook{para/before}{\bbbl@parabefore}%
6815   \AtBeginDocument{%
6816     \@ifpackageloaded{colortbl}%
6817     {\bbbl@replace\@tabular{$}{$%
6818       \def\bbbl@insidemath{0}%
6819       \def\bbbl@parabefore{\localerestoredirs}}%
6820     \bbbl@sreplace\@classz
6821     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6822     {}}}%
6823 \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.

```

6824 \AtBeginDocument{%
6825   \@ifpackageloaded{multicol}%

```

```

6826      {\toks\@expandafter\multi@column@out}%
6827      \edef\multi@column@out{\bodydir\pagedir\the\toks@}%
6828      }%
6829      \@ifpackageloaded{paracol}%
6830      {\edef\pcol@output{%
6831        \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6832      }%
6833 \fi
6834 \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`.

```

6835 \ifnum\bbl@bidimode>\z@ % Any bidi=
6836   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6837     \bbl@exp{%
6838       \mathdir\the\bodydir
6839       #1%           Once entered in math, set boxes to restore values
6840       \def\bbbl@insidemath{0}%
6841       \<ifmmode>%
6842       \everyvbox{%
6843         \the\everyvbox
6844         \bodydir\the\bodydir
6845         \mathdir\the\mathdir
6846         \everyhbox{\the\everyhbox}%
6847         \everyvbox{\the\everyvbox}}%
6848       \everyhbox{%
6849         \the\everyhbox
6850         \bodydir\the\bodydir
6851         \mathdir\the\mathdir
6852         \everyhbox{\the\everyhbox}%
6853         \everyvbox{\the\everyvbox}}%
6854       \<fi>}}%
6855   \def\@hangfrom#1{%
6856     \setbox\@tempboxa\hbox{#1}%
6857     \hangindent\wd\@tempboxa
6858     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6859       \shapemode\@ne
6860     \fi
6861     \noindent\box\@tempboxa}
6862 \fi
6863 \IfBabelLayout{tabular}
6864 {\let\bbl@OL@tabular\@tabular
6865  \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6866  \let\bbl@NL@tabular\@tabular
6867  \AtBeginDocument{%
6868    \ifx\bbl@NL@tabular\@tabular\else
6869      \bbl@exp{\in{\bbl@nextfake}{\@tabular}}%
6870      \ifin\else
6871        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6872      \fi
6873      \let\bbl@NL@tabular\@tabular
6874    \fi}}
6875 {}
6876 \IfBabelLayout{lists}
6877 {\let\bbl@OL@list\list
6878  \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6879  \let\bbl@NL@list\list
6880  \def\bbl@listparshape#1#2#3{%
6881    \parshape #1 #2 #3 %
6882    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6883      \shapemode\tw@

```

```

6884     \fi}}
6885   {}
6886 \IfBabelLayout{graphics}
6887 {\let\bbl@pictresetdir\relax
6888  \def\bbl@pictsetdir#1{%
6889    \ifcase\bbl@thetextdir
6890      \let\bbl@pictresetdir\relax
6891    \else
6892      \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6893        \or\textdir TLT
6894        \else\bodydir TLT \textdir TLT
6895      \fi
6896      % \(\text|par)dir required in pgf:
6897      \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6898    \fi}%
6899 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6900 \directlua{
6901   Babel.get_picture_dir = true
6902   Babel.picture_has_bidi = 0
6903   %
6904   function Babel.picture_dir (head)
6905     if not Babel.get_picture_dir then return head end
6906     if Babel.hlist_has_bidi(head) then
6907       Babel.picture_has_bidi = 1
6908     end
6909     return head
6910   end
6911   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6912     "Babel.picture_dir")
6913 }%
6914 \AtBeginDocument{%
6915   \def\LS@rot{%
6916     \setbox\@outputbox\vbox{%
6917       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6918   \long\def\put(#1,#2)#3{%
6919     \@killglue
6920     % Try:
6921     \ifx\bbl@pictresetdir\relax
6922       \def\bbl@tempc{0}%
6923     \else
6924       \directlua{
6925         Babel.get_picture_dir = true
6926         Babel.picture_has_bidi = 0
6927       }%
6928       \setbox\z@\hb@xt@\z@{%
6929         \@defaultunitsset\@tempdimc{#1}\unitlength
6930         \kern\@tempdimc
6931         #3\hss}% TODO: #3 executed twice (below). That's bad.
6932       \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6933     \fi
6934     % Do:
6935     \@defaultunitsset\@tempdimc{#2}\unitlength
6936     \raise\@tempdimc\hb@xt@\z@{%
6937       \@defaultunitsset\@tempdimc{#1}\unitlength
6938       \kern\@tempdimc
6939       {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6940     \ignorespaces}%
6941   \MakeRobust\put}%
6942 \AtBeginDocument
6943 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6944  \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6945    \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6946    \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%

```

```

6947      \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6948      \fi
6949      \ifx\tikzpicture\undefined\else
6950      \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6951      \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6952      \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6953      \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6954      \fi
6955      \ifx\tcolorbox\undefined\else
6956      \def\tcb@drawing@env@begin{%
6957      \csname tcb@before@\tcb@split@state\endcsname
6958      \bbl@pictsetdir\tw@
6959      \begin{\kvtcb@graphenv}%
6960      \tcb@bbdraw
6961      \tcb@apply@graph@patches}%
6962      \def\tcb@drawing@env@end{%
6963      \end{\kvtcb@graphenv}%
6964      \bbl@pictresetdir
6965      \csname tcb@after@\tcb@split@state\endcsname}%
6966      \fi
6967  }}
6968  {}

```

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.

```

6969 \IfBabelLayout{counters*}%
6970 {\bbl@add\bbl@opt@layout{.counters.}%
6971  \directlua{
6972    luatexbase.add_to_callback("process_output_buffer",
6973      Babel.discard_sublr , "Babel.discard_sublr") }%
6974  }}
6975 \IfBabelLayout{counters}%
6976 {\let\bbl@0L@@textsuperscript\@textsuperscript
6977  \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6978  \let\bbl@latinarabic=\@arabic
6979  \let\bbl@0L@@arabic\@arabic
6980  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6981  \ifpackagewith{babel}{bidi=default}%
6982  {\let\bbl@asciroman=\@roman
6983   \let\bbl@0L@@roman\@roman
6984   \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6985   \let\bbl@asciiRoman=\@Roman
6986   \let\bbl@0L@@roman\@Roman
6987   \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6988   \let\bbl@0L@labelenumii\labelenumii
6989   \def\labelenumii{\theenumii}%
6990   \let\bbl@0L@p@enumiii\p@enumiii
6991   \def\p@enumiii{\p@enumii}\theenumii{}}{}{}
6992 <@Footnote changes>
6993 \IfBabelLayout{footnotes}%
6994 {\let\bbl@0L@footnote\footnote
6995  \BabelFootnote\footnote\languagename{}}{}%
6996  \BabelFootnote\localfootnote\languagename{}}{}%
6997  \BabelFootnote\mainfootnote{}}{}{}
6998  {}

```

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.

```

6999 \IfBabelLayout{extras}%
7000 {\bbl@ncarg\let\bbl@0L@underline{underline }%
7001  \bbl@carg\bbl@sreplace{underline }%
7002   {\$@@@underline}{\bgroup\bbl@nextfake$@@@underline}%
7003  \bbl@carg\bbl@sreplace{underline }%

```

```

7004     {\m@th$}{\m@th$\egroup}%
7005     \let\bbl@0L@LaTeXe\LaTeXe
7006     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7007       \if b\expandafter\@car\@fseries\@nil\boldmath\fi
7008       \babelsublr{%
7009         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
7010   {}
7011 </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 discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

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

```

7012 <{*transforms}>
7013 Babel.linebreaking.replacements = {}
7014 Babel.linebreaking.replacements[0] = {} -- pre
7015 Babel.linebreaking.replacements[1] = {} -- post
7016
7017 function Babel.tovalue(v)
7018   if type(v) == 'table' then
7019     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7020   else
7021     return v
7022   end
7023 end
7024
7025 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7026
7027 function Babel.set_hboxed(head, gc)
7028   for item in node.traverse(head) do
7029     node.set_attribute(item, Babel.attr_hboxed, 1)
7030   end
7031   return head
7032 end
7033
7034 Babel.fetch_subtext = {}
7035
7036 Babel.ignore_pre_char = function(node)
7037   return (node.lang == Babel.nohyphenation)
7038 end
7039
7040 -- Merging both functions doesn't seem feasible, because there are too
7041 -- many differences.
7042 Babel.fetch_subtext[0] = function(head)
7043   local word_string = ''
7044   local word_nodes = {}
7045   local lang
7046   local item = head
7047   local inmath = false
7048
7049   while item do
7050     if item.id == 11 then
7051       inmath = (item.subtype == 0)

```



```

7053     end
7054
7055     if inmath then
7056         -- pass
7057
7058     elseif item.id == 29 then
7059         local locale = node.get_attribute(item, Babel.attr_locale)
7060
7061         if lang == locale or lang == nil then
7062             lang = lang or locale
7063             if Babel.ignore_pre_char(item) then
7064                 word_string = word_string .. Babel.us_char
7065             else
7066                 if node.has_attribute(item, Babel.attr_hboxed) then
7067                     word_string = word_string .. Babel.us_char
7068                 else
7069                     word_string = word_string .. unicode.utf8.char(item.char)
7070                 end
7071             end
7072             word_nodes[#word_nodes+1] = item
7073         else
7074             break
7075         end
7076
7077     elseif item.id == 12 and item.subtype == 13 then
7078         if node.has_attribute(item, Babel.attr_hboxed) then
7079             word_string = word_string .. Babel.us_char
7080         else
7081             word_string = word_string .. ' '
7082         end
7083         word_nodes[#word_nodes+1] = item
7084
7085         -- Ignore leading unrecognized nodes, too.
7086     elseif word_string ~= '' then
7087         word_string = word_string .. Babel.us_char
7088         word_nodes[#word_nodes+1] = item -- Will be ignored
7089     end
7090
7091     item = item.next
7092 end
7093
7094 -- Here and above we remove some trailing chars but not the
7095 -- corresponding nodes. But they aren't accessed.
7096 if word_string:sub(-1) == ' ' then
7097     word_string = word_string:sub(1,-2)
7098 end
7099 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7100 return word_string, word_nodes, item, lang
7101 end
7102
7103 Babel.fetch_subtext[1] = function(head)
7104     local word_string = ''
7105     local word_nodes = {}
7106     local lang
7107     local item = head
7108     local inmath = false
7109
7110     while item do
7111
7112         if item.id == 11 then
7113             inmath = (item.subtype == 0)
7114         end
7115

```

```

7116     if inmath then
7117         -- pass
7118
7119     elseif item.id == 29 then
7120         if item.lang == lang or lang == nil then
7121             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7122                 lang = lang or item.lang
7123                 if node.has_attribute(item, Babel.attr_hboxed) then
7124                     word_string = word_string .. Babel.us_char
7125                 else
7126                     word_string = word_string .. unicode.utf8.char(item.char)
7127                 end
7128                 word_nodes[#word_nodes+1] = item
7129             end
7130         else
7131             break
7132         end
7133
7134     elseif item.id == 7 and item.subtype == 2 then
7135         if node.has_attribute(item, Babel.attr_hboxed) then
7136             word_string = word_string .. Babel.us_char
7137         else
7138             word_string = word_string .. '='
7139         end
7140         word_nodes[#word_nodes+1] = item
7141
7142     elseif item.id == 7 and item.subtype == 3 then
7143         if node.has_attribute(item, Babel.attr_hboxed) then
7144             word_string = word_string .. Babel.us_char
7145         else
7146             word_string = word_string .. '|'
7147         end
7148         word_nodes[#word_nodes+1] = item
7149
7150         -- (1) Go to next word if nothing was found, and (2) implicitly
7151         -- remove leading USs.
7152     elseif word_string == '' then
7153         -- pass
7154
7155         -- This is the responsible for splitting by words.
7156     elseif (item.id == 12 and item.subtype == 13) then
7157         break
7158
7159     else
7160         word_string = word_string .. Babel.us_char
7161         word_nodes[#word_nodes+1] = item -- Will be ignored
7162     end
7163
7164     item = item.next
7165 end
7166
7167 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7168 return word_string, word_nodes, item, lang
7169 end
7170
7171 function Babel.pre_hyphenate_replace(head)
7172     Babel.hyphenate_replace(head, 0)
7173 end
7174
7175 function Babel.post_hyphenate_replace(head)
7176     Babel.hyphenate_replace(head, 1)
7177 end
7178

```

```

7179 Babel.us_char = string.char(31)
7180
7181 function Babel.hyphenate_replace(head, mode)
7182     local u = unicode.utf8
7183     local lbkr = Babel.linebreaking.replacements[mode]
7184     local tovalue = Babel.tovalue
7185
7186     local word_head = head
7187
7188     while true do -- for each subtext block
7189
7190         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7191
7192         if Babel.debug then
7193             print()
7194             print((mode == 0) and '@@@<' or '@@@>', w)
7195         end
7196
7197         if nw == nil and w == '' then break end
7198
7199         if not lang then goto next end
7200         if not lbkr[lang] then goto next end
7201
7202         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7203         -- loops are nested.
7204         for k=1, #lbkr[lang] do
7205             local p = lbkr[lang][k].pattern
7206             local r = lbkr[lang][k].replace
7207             local attr = lbkr[lang][k].attr or -1
7208
7209             if Babel.debug then
7210                 print('*****', p, mode)
7211             end
7212
7213             -- This variable is set in some cases below to the first *byte*
7214             -- after the match, either as found by u.match (faster) or the
7215             -- computed position based on sc if w has changed.
7216             local last_match = 0
7217             local step = 0
7218
7219             -- For every match.
7220             while true do
7221                 if Babel.debug then
7222                     print('====')
7223                 end
7224                 local new -- used when inserting and removing nodes
7225                 local dummy_node -- used by after
7226
7227                 local matches = { u.match(w, p, last_match) }
7228
7229                 if #matches < 2 then break end
7230
7231                 -- Get and remove empty captures (with ()'s, which return a
7232                 -- number with the position), and keep actual captures
7233                 -- (from (...)), if any, in matches.
7234                 local first = table.remove(matches, 1)
7235                 local last = table.remove(matches, #matches)
7236                 -- Non re-fetched substrings may contain \31, which separates
7237                 -- subsubstrings.
7238                 if string.find(w:sub(first, last-1), Babel.us_char) then break end
7239
7240                 local save_last = last -- with A()BC()D, points to D
7241

```

```

7242     -- Fix offsets, from bytes to unicode. Explained above.
7243     first = u.len(w:sub(1, first-1)) + 1
7244     last  = u.len(w:sub(1, last-1)) -- now last points to C
7245
7246     -- This loop stores in a small table the nodes
7247     -- corresponding to the pattern. Used by 'data' to provide a
7248     -- predictable behavior with 'insert' (w_nodes is modified on
7249     -- the fly), and also access to 'remove'd nodes.
7250     local sc = first-1          -- Used below, too
7251     local data_nodes = {}
7252
7253     local enabled = true
7254     for q = 1, last-first+1 do
7255         data_nodes[q] = w_nodes[sc+q]
7256         if enabled
7257             and attr > -1
7258             and not node.has_attribute(data_nodes[q], attr)
7259         then
7260             enabled = false
7261         end
7262     end
7263
7264     -- This loop traverses the matched substring and takes the
7265     -- corresponding action stored in the replacement list.
7266     -- sc = the position in substr nodes / string
7267     -- rc = the replacement table index
7268     local rc = 0
7269
7270     ----- TODO. dummy_node?
7271     while rc < last-first+1 or dummy_node do -- for each replacement
7272         if Babel.debug then
7273             print('.....', rc + 1)
7274         end
7275         sc = sc + 1
7276         rc = rc + 1
7277
7278         if Babel.debug then
7279             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7280             local ss = ''
7281             for itt in node.traverse(head) do
7282                 if itt.id == 29 then
7283                     ss = ss .. unicode.utf8.char(itt.char)
7284                 else
7285                     ss = ss .. '{' .. itt.id .. '}'
7286                 end
7287             end
7288             print('*****', ss)
7289         end
7290
7291         local crep = r[rc]
7292         local item = w_nodes[sc]
7293         local item_base = item
7294         local placeholder = Babel.us_char
7295         local d
7296
7297         if crep and crep.data then
7298             item_base = data_nodes[crep.data]
7299         end
7300
7301         if crep then
7302             step = crep.step or step
7303         end
7304     end

```

```

7305
7306     if crep and crep.after then
7307         crep.insert = true
7308         if dummy_node then
7309             item = dummy_node
7310         else -- TODO. if there is a node after?
7311             d = node.copy(item_base)
7312             head, item = node.insert_after(head, item, d)
7313             dummy_node = item
7314         end
7315     end
7316
7317     if crep and not crep.after and dummy_node then
7318         node.remove(head, dummy_node)
7319         dummy_node = nil
7320     end
7321
7322     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7323         if step == 0 then
7324             last_match = save_last    -- Optimization
7325         else
7326             last_match = utf8.offset(w, sc+step)
7327         end
7328         goto next
7329
7330     elseif crep == nil or crep.remove then
7331         node.remove(head, item)
7332         table.remove(w_nodes, sc)
7333         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7334         sc = sc - 1 -- Nothing has been inserted.
7335         last_match = utf8.offset(w, sc+1+step)
7336         goto next
7337
7338     elseif crep and crep.kashida then -- Experimental
7339         node.set_attribute(item,
7340             Babel.attr_kashida,
7341             crep.kashida)
7342         last_match = utf8.offset(w, sc+1+step)
7343         goto next
7344
7345     elseif crep and crep.string then
7346         local str = crep.string(matches)
7347         if str == '' then -- Gather with nil
7348             node.remove(head, item)
7349             table.remove(w_nodes, sc)
7350             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7351             sc = sc - 1 -- Nothing has been inserted.
7352         else
7353             local loop_first = true
7354             for s in string.utfvalues(str) do
7355                 d = node.copy(item_base)
7356                 d.char = s
7357                 if loop_first then
7358                     loop_first = false
7359                     head, new = node.insert_before(head, item, d)
7360                     if sc == 1 then
7361                         word_head = head
7362                     end
7363                     w_nodes[sc] = d
7364                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7365                 else
7366                     sc = sc + 1
7367                     head, new = node.insert_before(head, item, d)

```

```

7368         table.insert(w_nodes, sc, new)
7369         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7370     end
7371     if Babel.debug then
7372         print('.....', 'str')
7373         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7374     end
7375     end -- for
7376     node.remove(head, item)
7377 end -- if ''
7378 last_match = utf8.offset(w, sc+1+step)
7379 goto next
7380
7381 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7382     d = node.new(7, 3) -- (disc, regular)
7383     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7384     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7385     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7386     d.attr = item_base.attr
7387     if crep.pre == nil then -- TeXbook p96
7388         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7389     else
7390         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7391     end
7392     placeholder = '|'
7393     head, new = node.insert_before(head, item, d)
7394
7395 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7396     -- ERROR
7397
7398 elseif crep and crep.penalty then
7399     d = node.new(14, 0) -- (penalty, userpenalty)
7400     d.attr = item_base.attr
7401     d.penalty = tovalue(crep.penalty)
7402     head, new = node.insert_before(head, item, d)
7403
7404 elseif crep and crep.space then
7405     -- 655360 = 10 pt = 10 * 65536 sp
7406     d = node.new(12, 13) -- (glue, spaceskip)
7407     local quad = font.getfont(item_base.font).size or 655360
7408     node.setglue(d, tovalue(crep.space[1]) * quad,
7409                  tovalue(crep.space[2]) * quad,
7410                  tovalue(crep.space[3]) * quad)
7411     if mode == 0 then
7412         placeholder = ' '
7413     end
7414     head, new = node.insert_before(head, item, d)
7415
7416 elseif crep and crep.norule then
7417     -- 655360 = 10 pt = 10 * 65536 sp
7418     d = node.new(2, 3) -- (rule, empty) = \no*rule
7419     local quad = font.getfont(item_base.font).size or 655360
7420     d.width = tovalue(crep.norule[1]) * quad
7421     d.height = tovalue(crep.norule[2]) * quad
7422     d.depth = tovalue(crep.norule[3]) * quad
7423     head, new = node.insert_before(head, item, d)
7424
7425 elseif crep and crep.spacefactor then
7426     d = node.new(12, 13) -- (glue, spaceskip)
7427     local base_font = font.getfont(item_base.font)
7428     node.setglue(d,
7429                  tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7430                  tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],

```

```

7431         tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7432     if mode == 0 then
7433         placeholder = ' '
7434     end
7435     head, new = node.insert_before(head, item, d)
7436
7437 elseif mode == 0 and crep and crep.space then
7438     -- ERROR
7439
7440 elseif crep and crep.kern then
7441     d = node.new(13, 1)      -- (kern, user)
7442     local quad = font.getfont(item_base.font).size or 655360
7443     d.attr = item_base.attr
7444     d.kern = tovalue(crep.kern) * quad
7445     head, new = node.insert_before(head, item, d)
7446
7447 elseif crep and crep.node then
7448     d = node.new(crep.node[1], crep.node[2])
7449     d.attr = item_base.attr
7450     head, new = node.insert_before(head, item, d)
7451
7452 end -- i.e., replacement cases
7453
7454 -- Shared by disc, space(factor), kern, node and penalty.
7455 if sc == 1 then
7456     word_head = head
7457 end
7458 if crep.insert then
7459     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7460     table.insert(w_nodes, sc, new)
7461     last = last + 1
7462 else
7463     w_nodes[sc] = d
7464     node.remove(head, item)
7465     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7466 end
7467
7468 last_match = utf8.offset(w, sc+1+step)
7469
7470 ::next::
7471
7472 end -- for each replacement
7473
7474 if Babel.debug then
7475     print('.....', '/')
7476     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7477 end
7478
7479 if dummy_node then
7480     node.remove(head, dummy_node)
7481     dummy_node = nil
7482 end
7483
7484 end -- for match
7485
7486 end -- for patterns
7487
7488 ::next::
7489 word_head = nw
7490 end -- for substring
7491 return head
7492 end
7493

```

```

7494 -- This table stores capture maps, numbered consecutively
7495 Babel.capture_maps = {}
7496
7497 -- The following functions belong to the next macro
7498 function Babel.capture_func(key, cap)
7499   local ret = "[" .. cap:gsub('{{[0-9]}}', "")..m[%1]..["] .. "]"
7500   local cnt
7501   local u = unicode.utf8
7502   ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7503   if cnt == 0 then
7504     ret = u.gsub(ret, '{(%x%x%x%x+)}',
7505       function (n)
7506         return u.char(tonumber(n, 16))
7507       end)
7508   end
7509   ret = ret:gsub("%[%]%.%", '')
7510   ret = ret:gsub("%.%.%[%]%", '')
7511   return key .. "[=function(m) return ]] .. ret .. [[ end]]
7512 end
7513
7514 function Babel.capt_map(from, mapno)
7515   return Babel.capture_maps[mapno][from] or from
7516 end
7517
7518 -- Handle the {n|abc|ABC} syntax in captures
7519 function Babel.capture_func_map(capno, from, to)
7520   local u = unicode.utf8
7521   from = u.gsub(from, '{(%x%x%x%x+)}',
7522     function (n)
7523       return u.char(tonumber(n, 16))
7524     end)
7525   to = u.gsub(to, '{(%x%x%x%x+)}',
7526     function (n)
7527       return u.char(tonumber(n, 16))
7528     end)
7529   local froms = {}
7530   for s in string.utfcharacters(from) do
7531     table.insert(froms, s)
7532   end
7533   local cnt = 1
7534   table.insert(Babel.capture_maps, {})
7535   local mlen = table.getn(Babel.capture_maps)
7536   for s in string.utfcharacters(to) do
7537     Babel.capture_maps[mlen][froms[cnt]] = s
7538     cnt = cnt + 1
7539   end
7540   return "]"..Babel.capt_map(m[" .. capno .. "], " ..
7541     (mlen) .. ").." .. "["
7542 end
7543
7544 -- Create/Extend reversed sorted list of kashida weights:
7545 function Babel.capture_kashida(key, wt)
7546   wt = tonumber(wt)
7547   if Babel.kashida_wts then
7548     for p, q in ipairs(Babel.kashida_wts) do
7549       if wt == q then
7550         break
7551       elseif wt > q then
7552         table.insert(Babel.kashida_wts, p, wt)
7553         break
7554       elseif table.getn(Babel.kashida_wts) == p then
7555         table.insert(Babel.kashida_wts, wt)
7556       end

```



```

7557     end
7558 else
7559     Babel.kashida_wts = { wt }
7560 end
7561 return 'kashida = ' .. wt
7562 end
7563
7564 function Babel.capture_node(id, subtype)
7565     local sbt = 0
7566     for k, v in pairs(node.subtypes(id)) do
7567         if v == subtype then sbt = k end
7568     end
7569     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7570 end
7571
7572 -- Experimental: applies prehyphenation transforms to a string (letters
7573 -- and spaces).
7574 function Babel.string_prehyphenation(str, locale)
7575     local n, head, last, res
7576     head = node.new(8, 0) -- dummy (hack just to start)
7577     last = head
7578     for s in string.utfvalues(str) do
7579         if s == 20 then
7580             n = node.new(12, 0)
7581         else
7582             n = node.new(29, 0)
7583             n.char = s
7584         end
7585         node.set_attribute(n, Babel.attr_locale, locale)
7586         last.next = n
7587         last = n
7588     end
7589     head = Babel.hyphenate_replace(head, 0)
7590     res = ''
7591     for n in node.traverse(head) do
7592         if n.id == 12 then
7593             res = res .. ' '
7594         elseif n.id == 29 then
7595             res = res .. unicode.utf8.char(n.char)
7596         end
7597     end
7598     tex.print(res)
7599 end
7600 </transforms>

```

10.14 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is

still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

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

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

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

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

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

```
7601 (*basic-r)
7602 Babel.bidi_enabled = true
7603
7604 require('babel-data-bidi.lua')
7605
7606 local characters = Babel.characters
7607 local ranges = Babel.ranges
7608
7609 local DIR = node.id("dir")
7610
7611 local function dir_mark(head, from, to, outer)
7612   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7613   local d = node.new(DIR)
7614   d.dir = '+' .. dir
7615   node.insert_before(head, from, d)
7616   d = node.new(DIR)
7617   d.dir = '-' .. dir
7618   node.insert_after(head, to, d)
7619 end
7620
7621 function Babel.bidi(head, ispar)
7622   local first_n, last_n          -- first and last char with nums
7623   local last_es                  -- an auxiliary 'last' used with nums
7624   local first_d, last_d          -- first and last char in L/R block
7625   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):

```
7626   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7627   local strong_lr = (strong == 'l') and 'l' or 'r'
7628   local outer = strong
7629
7630   local new_dir = false
7631   local first_dir = false
7632   local inmath = false
7633
7634   local last_lr
7635
7636   local type_n = ''
7637
7638   for item in node.traverse(head) do
7639
```

```

7640 -- three cases: glyph, dir, otherwise
7641 if item.id == node.id'glyph'
7642   or (item.id == 7 and item.subtype == 2) then
7643
7644   local itemchar
7645   if item.id == 7 and item.subtype == 2 then
7646     itemchar = item.replace.char
7647   else
7648     itemchar = item.char
7649   end
7650   local chardata = characters[itemchar]
7651   dir = chardata and chardata.d or nil
7652   if not dir then
7653     for nn, et in ipairs(ranges) do
7654       if itemchar < et[1] then
7655         break
7656       elseif itemchar <= et[2] then
7657         dir = et[3]
7658         break
7659       end
7660     end
7661   end
7662   dir = dir or 'l'
7663   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).

```

7664   if new_dir then
7665     attr_dir = 0
7666     for at in node.traverse(item.attr) do
7667       if at.number == Babel.attr_dir then
7668         attr_dir = at.value & 0x3
7669       end
7670     end
7671     if attr_dir == 1 then
7672       strong = 'r'
7673     elseif attr_dir == 2 then
7674       strong = 'al'
7675     else
7676       strong = 'l'
7677     end
7678     strong_lr = (strong == 'l') and 'l' or 'r'
7679     outer = strong_lr
7680     new_dir = false
7681   end
7682
7683   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7684   dir_real = dir -- We need dir_real to set strong below
7685   if dir == 'al' then dir = 'r' end -- W3

```

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

```

7686   if strong == 'al' then
7687     if dir == 'en' then dir = 'an' end -- W2
7688     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7689     strong_lr = 'r' -- W3
7690   end

```

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

```

7691 elseif item.id == node.id'dir' and not inmath then
7692   new_dir = true
7693   dir = nil
7694 elseif item.id == node.id'math' then
7695   inmath = (item.subtype == 0)
7696 else
7697   dir = nil          -- Not a char
7698 end

```

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

```

7699 if dir == 'en' or dir == 'an' or dir == 'et' then
7700   if dir ~= 'et' then
7701     type_n = dir
7702   end
7703   first_n = first_n or item
7704   last_n = last_es or item
7705   last_es = nil
7706 elseif dir == 'es' and last_n then -- W3+W6
7707   last_es = item
7708 elseif dir == 'cs' then          -- it's right - do nothing
7709 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7710   if strong_lr == 'r' and type_n ~= '' then
7711     dir_mark(head, first_n, last_n, 'r')
7712   elseif strong_lr == 'l' and first_d and type_n == 'an' then
7713     dir_mark(head, first_n, last_n, 'r')
7714     dir_mark(head, first_d, last_d, outer)
7715     first_d, last_d = nil, nil
7716   elseif strong_lr == 'l' and type_n ~= '' then
7717     last_d = last_n
7718   end
7719   type_n = ''
7720   first_n, last_n = nil, nil
7721 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:

```

7722 if dir == 'l' or dir == 'r' then
7723   if dir ~= outer then
7724     first_d = first_d or item
7725     last_d = item
7726   elseif first_d and dir ~= strong_lr then
7727     dir_mark(head, first_d, last_d, outer)
7728     first_d, last_d = nil, nil
7729   end
7730 end

```

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

```

7731 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7732   item.char = characters[item.char] and
7733     characters[item.char].m or item.char
7734 elseif (dir or new_dir) and last_lr ~= item then
7735   local mir = outer .. strong_lr .. (dir or outer)
7736   if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7737     for ch in node.traverse(node.next(last_lr)) do

```

```

7738         if ch == item then break end
7739         if ch.id == node.id'glyph' and characters[ch.char] then
7740             ch.char = characters[ch.char].m or ch.char
7741         end
7742     end
7743 end
7744 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).

```

7745     if dir == 'l' or dir == 'r' then
7746         last_lr = item
7747         strong = dir_real          -- Don't search back - best save now
7748         strong_lr = (strong == 'l') and 'l' or 'r'
7749     elseif new_dir then
7750         last_lr = nil
7751     end
7752 end

```

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

```

7753     if last_lr and outer == 'r' then
7754         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7755             if characters[ch.char] then
7756                 ch.char = characters[ch.char].m or ch.char
7757             end
7758         end
7759     end
7760     if first_n then
7761         dir_mark(head, first_n, last_n, outer)
7762     end
7763     if first_d then
7764         dir_mark(head, first_d, last_d, outer)
7765     end

```

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

```

7766     return node.prev(head) or head
7767 end
7768 </basic-r>

```

And here the Lua code for bidi=basic:

```

7769 <(*basic>
7770 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7771
7772 Babel.fontmap = Babel.fontmap or {}
7773 Babel.fontmap[0] = {}          -- l
7774 Babel.fontmap[1] = {}          -- r
7775 Babel.fontmap[2] = {}          -- al/an
7776
7777 -- To cancel mirroring. Also OML, OMS, U?
7778 Babel.symbol_fonts = Babel.symbol_fonts or {}
7779 Babel.symbol_fonts[font.id('tenln')] = true
7780 Babel.symbol_fonts[font.id('tenlnw')] = true
7781 Babel.symbol_fonts[font.id('tencirc')] = true
7782 Babel.symbol_fonts[font.id('tencircw')] = true
7783
7784 Babel.bidi_enabled = true
7785 Babel.mirroring_enabled = true
7786
7787 require('babel-data-bidi.lua')
7788
7789 local characters = Babel.characters
7790 local ranges = Babel.ranges
7791

```

```

7792 local DIR = node.id('dir')
7793 local GLYPH = node.id('glyph')
7794
7795 local function insert_implicit(head, state, outer)
7796   local new_state = state
7797   if state.sim and state.eim and state.sim ~= state.eim then
7798     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7799     local d = node.new(DIR)
7800     d.dir = '+' .. dir
7801     node.insert_before(head, state.sim, d)
7802     local d = node.new(DIR)
7803     d.dir = '-' .. dir
7804     node.insert_after(head, state.eim, d)
7805   end
7806   new_state.sim, new_state.eim = nil, nil
7807   return head, new_state
7808 end
7809
7810 local function insert_numeric(head, state)
7811   local new
7812   local new_state = state
7813   if state.san and state.ean and state.san ~= state.ean then
7814     local d = node.new(DIR)
7815     d.dir = '+TLT'
7816     _, new = node.insert_before(head, state.san, d)
7817     if state.san == state.sim then state.sim = new end
7818     local d = node.new(DIR)
7819     d.dir = '-TLT'
7820     _, new = node.insert_after(head, state.ean, d)
7821     if state.ean == state.eim then state.eim = new end
7822   end
7823   new_state.san, new_state.ean = nil, nil
7824   return head, new_state
7825 end
7826
7827 local function glyph_not_symbol_font(node)
7828   if node.id == GLYPH then
7829     return not Babel.symbol_fonts[node.font]
7830   else
7831     return false
7832   end
7833 end
7834
7835 -- TODO - \hbox with an explicit dir can lead to wrong results
7836 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7837 -- was made to improve the situation, but the problem is the 3-dir
7838 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7839 -- well.
7840
7841 function Babel.bidi(head, ispar, hdir)
7842   local d -- d is used mainly for computations in a loop
7843   local prev_d = ''
7844   local new_d = false
7845
7846   local nodes = {}
7847   local outer_first = nil
7848   local inmath = false
7849
7850   local glue_d = nil
7851   local glue_i = nil
7852
7853   local has_en = false
7854   local first_et = nil

```

```

7855
7856 local has_hyperlink = false
7857
7858 local ATDIR = Babel.attr_dir
7859 local attr_d
7860
7861 local save_outer
7862 local temp = node.get_attribute(head, ATDIR)
7863 if temp then
7864     temp = temp & 0x3
7865     save_outer = (temp == 0 and 'l') or
7866                 (temp == 1 and 'r') or
7867                 (temp == 2 and 'al')
7868 elseif ispar then -- Or error? Shouldn't happen
7869     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7870 else -- Or error? Shouldn't happen
7871     save_outer = ('TRT' == hdir) and 'r' or 'l'
7872 end
7873 -- when the callback is called, we are just _after_ the box,
7874 -- and the textdir is that of the surrounding text
7875 -- if not ispar and hdir ~= tex.textdir then
7876 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7877 -- end
7878 local outer = save_outer
7879 local last = outer
7880 -- 'al' is only taken into account in the first, current loop
7881 if save_outer == 'al' then save_outer = 'r' end
7882
7883 local fontmap = Babel.fontmap
7884
7885 for item in node.traverse(head) do
7886
7887     -- In what follows, #node is the last (previous) node, because the
7888     -- current one is not added until we start processing the neutrals.
7889
7890     -- three cases: glyph, dir, otherwise
7891     if glyph_not_symbol_font(item)
7892         or (item.id == 7 and item.subtype == 2) then
7893
7894         if node.get_attribute(item, ATDIR) == 128 then goto nextnode end
7895
7896         local d_font = nil
7897         local item_r
7898         if item.id == 7 and item.subtype == 2 then
7899             item_r = item.replace -- automatic discs have just 1 glyph
7900         else
7901             item_r = item
7902         end
7903
7904         local chardata = characters[item_r.char]
7905         d = chardata and chardata.d or nil
7906         if not d or d == 'nsm' then
7907             for nn, et in ipairs(ranges) do
7908                 if item_r.char < et[1] then
7909                     break
7910                 elseif item_r.char <= et[2] then
7911                     if not d then d = et[3]
7912                     elseif d == 'nsm' then d_font = et[3]
7913                 end
7914                 break
7915             end
7916         end
7917     end

```

```

7918     d = d or 'l'
7919
7920     -- A short 'pause' in bidi for mapfont
7921     d_font = d_font or d
7922     d_font = (d_font == 'l' and 0) or
7923               (d_font == 'nsm' and 0) or
7924               (d_font == 'r' and 1) or
7925               (d_font == 'al' and 2) or
7926               (d_font == 'an' and 2) or nil
7927     if d_font and fontmap and fontmap[d_font][item_r.font] then
7928         item_r.font = fontmap[d_font][item_r.font]
7929     end
7930
7931     if new_d then
7932         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7933         if inmath then
7934             attr_d = 0
7935         else
7936             attr_d = node.get_attribute(item, ATDIR)
7937             attr_d = attr_d & 0x3
7938         end
7939         if attr_d == 1 then
7940             outer_first = 'r'
7941             last = 'r'
7942         elseif attr_d == 2 then
7943             outer_first = 'r'
7944             last = 'al'
7945         else
7946             outer_first = 'l'
7947             last = 'l'
7948         end
7949         outer = last
7950         has_en = false
7951         first_et = nil
7952         new_d = false
7953     end
7954
7955     if glue_d then
7956         if (d == 'l' and 'l' or 'r') ~= glue_d then
7957             table.insert(nodes, {glue_i, 'on', nil})
7958         end
7959         glue_d = nil
7960         glue_i = nil
7961     end
7962
7963     elseif item.id == DIR then
7964         d = nil
7965
7966         if head ~= item then new_d = true end
7967
7968     elseif item.id == node.id'glue' and item.subtype == 13 then
7969         glue_d = d
7970         glue_i = item
7971         d = nil
7972
7973     elseif item.id == node.id'math' then
7974         inmath = (item.subtype == 0)
7975
7976     elseif item.id == 8 and item.subtype == 19 then
7977         has_hyperlink = true
7978
7979     else
7980         d = nil

```



```

7981 end
7982
7983 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7984 if last == 'al' and d == 'en' then
7985     d = 'an'           -- W3
7986 elseif last == 'al' and (d == 'et' or d == 'es') then
7987     d = 'on'           -- W6
7988 end
7989
7990 -- EN + CS/ES + EN      -- W4
7991 if d == 'en' and #nodes >= 2 then
7992     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7993         and nodes[#nodes-1][2] == 'en' then
7994         nodes[#nodes][2] = 'en'
7995     end
7996 end
7997
7998 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7999 if d == 'an' and #nodes >= 2 then
8000     if (nodes[#nodes][2] == 'cs')
8001         and nodes[#nodes-1][2] == 'an' then
8002         nodes[#nodes][2] = 'an'
8003     end
8004 end
8005
8006 -- ET/EN                -- W5 + W7->l / W6->on
8007 if d == 'et' then
8008     first_et = first_et or (#nodes + 1)
8009 elseif d == 'en' then
8010     has_en = true
8011     first_et = first_et or (#nodes + 1)
8012 elseif first_et then      -- d may be nil here !
8013     if has_en then
8014         if last == 'l' then
8015             temp = 'l'    -- W7
8016         else
8017             temp = 'en'   -- W5
8018         end
8019     else
8020         temp = 'on'       -- W6
8021     end
8022     for e = first_et, #nodes do
8023         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8024     end
8025     first_et = nil
8026     has_en = false
8027 end
8028
8029 -- Force mathdir in math if ON (currently works as expected only
8030 -- with 'l')
8031
8032 if inmath and d == 'on' then
8033     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8034 end
8035
8036 if d then
8037     if d == 'al' then
8038         d = 'r'
8039         last = 'al'
8040     elseif d == 'l' or d == 'r' then
8041         last = d
8042     end
8043     prev_d = d

```

```

8044     table.insert(nodes, {item, d, outer_first})
8045 end
8046
8047 node.set_attribute(item, ATDIR, 128)
8048 outer_first = nil
8049
8050 ::nextnode::
8051
8052 end -- for each node
8053
8054 -- TODO -- repeated here in case EN/ET is the last node. Find a
8055 -- better way of doing things:
8056 if first_et then      -- dir may be nil here !
8057     if has_en then
8058         if last == 'l' then
8059             temp = 'l'    -- W7
8060         else
8061             temp = 'en'    -- W5
8062         end
8063     else
8064         temp = 'on'    -- W6
8065     end
8066     for e = first_et, #nodes do
8067         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8068     end
8069 end
8070
8071 -- dummy node, to close things
8072 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8073
8074 ----- NEUTRAL -----
8075
8076 outer = save_outer
8077 last = outer
8078
8079 local first_on = nil
8080
8081 for q = 1, #nodes do
8082     local item
8083
8084     local outer_first = nodes[q][3]
8085     outer = outer_first or outer
8086     last = outer_first or last
8087
8088     local d = nodes[q][2]
8089     if d == 'an' or d == 'en' then d = 'r' end
8090     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8091
8092     if d == 'on' then
8093         first_on = first_on or q
8094     elseif first_on then
8095         if last == d then
8096             temp = d
8097         else
8098             temp = outer
8099         end
8100         for r = first_on, q - 1 do
8101             nodes[r][2] = temp
8102             item = nodes[r][1]    -- MIRRORING
8103             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8104                 and temp == 'r' and characters[item.char] then
8105                 local font_mode = ''
8106                 if item.font > 0 and font.fonts[item.font].properties then

```

```

8107         font_mode = font.fonts[item.font].properties.mode
8108     end
8109     if font_mode ~= 'harf' and font_mode ~= 'plug' then
8110         item.char = characters[item.char].m or item.char
8111     end
8112 end
8113 end
8114     first_on = nil
8115 end
8116
8117     if d == 'r' or d == 'l' then last = d end
8118 end
8119
8120 ----- IMPLICIT, REORDER -----
8121
8122 outer = save_outer
8123 last = outer
8124
8125 local state = {}
8126 state.has_r = false
8127
8128 for q = 1, #nodes do
8129
8130     local item = nodes[q][1]
8131
8132     outer = nodes[q][3] or outer
8133
8134     local d = nodes[q][2]
8135
8136     if d == 'nsm' then d = last end          -- W1
8137     if d == 'en' then d = 'an' end
8138     local isdir = (d == 'r' or d == 'l')
8139
8140     if outer == 'l' and d == 'an' then
8141         state.san = state.san or item
8142         state.ean = item
8143     elseif state.san then
8144         head, state = insert_numeric(head, state)
8145     end
8146
8147     if outer == 'l' then
8148         if d == 'an' or d == 'r' then      -- im -> implicit
8149             if d == 'r' then state.has_r = true end
8150             state.sim = state.sim or item
8151             state.eim = item
8152         elseif d == 'l' and state.sim and state.has_r then
8153             head, state = insert_implicit(head, state, outer)
8154         elseif d == 'l' then
8155             state.sim, state.eim, state.has_r = nil, nil, false
8156         end
8157     else
8158         if d == 'an' or d == 'l' then
8159             if nodes[q][3] then -- nil except after an explicit dir
8160                 state.sim = item -- so we move sim 'inside' the group
8161             else
8162                 state.sim = state.sim or item
8163             end
8164             state.eim = item
8165         elseif d == 'r' and state.sim then
8166             head, state = insert_implicit(head, state, outer)
8167         elseif d == 'r' then
8168             state.sim, state.eim = nil, nil
8169         end
8170     end
8171 end

```

```

8170     end
8171
8172     if isdir then
8173         last = d -- Don't search back - best save now
8174     elseif d == 'on' and state.san then
8175         state.san = state.san or item
8176         state.ean = item
8177     end
8178
8179 end
8180
8181 head = node.prev(head) or head
8182
8183 ----- FIX HYPERLINKS -----
8184
8185 if has_hyperlink then
8186     local flag, linking = 0, 0
8187     for item in node.traverse(head) do
8188         if item.id == DIR then
8189             if item.dir == '+TRT' or item.dir == '+TLT' then
8190                 flag = flag + 1
8191             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8192                 flag = flag - 1
8193             end
8194             elseif item.id == 8 and item.subtype == 19 then
8195                 linking = flag
8196             elseif item.id == 8 and item.subtype == 20 then
8197                 if linking > 0 then
8198                     if item.prev.id == DIR and
8199                        (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8200                         d = node.new(DIR)
8201                         d.dir = item.prev.dir
8202                         node.remove(head, item.prev)
8203                         node.insert_after(head, item, d)
8204                     end
8205                 end
8206                 linking = 0
8207             end
8208         end
8209     end
8210
8211 ----- FIX PENALTIES -----
8212
8213 for item in node.traverse_id(10, head) do
8214     local p = item
8215     local flag = false
8216     while p.prev and p.prev.id == 14 do
8217         flag = true
8218         p = p.prev
8219     end
8220     if flag then
8221         node.insert_before(head, p, node.copy(item))
8222         node.remove(head, item)
8223     end
8224 end
8225
8226 return head
8227 end
8228 -- Make sure anything is marked as 'bidi done' (including nodes inserted
8229 -- after the babel algorithm). 128 = 1000 0000.
8230 function Babel.unset_atdir(head)
8231     local ATDIR = Babel.attr_dir
8232     for item in node.traverse(head) do

```

```

8233     node.set_attribute(item, ATDIR, 128)
8234 end
8235 return head
8236 end
8237 </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.

```

8238 <{*nil}>
8239 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8240 \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.

```

8241 \ifx\l@nil\undefined
8242   \newlanguage\l@nil
8243   \namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
8244   \let\bbl@elt\relax
8245   \edef\bbl@languages{% Add it to the list of languages
8246     \bbl@languages\bbl@elt{nil}{\the\l@nil}\{}}
8247 \fi

```

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

```

8248 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

`\captionnil`

`\datenil`

```

8249 \let\captionnil\@empty
8250 \let\datenil\@empty

```

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

```

8251 \def\bbl@inidata@nil{%
8252   \bbl@elt{identification}{tag.ini}{und}%
8253   \bbl@elt{identification}{load.level}{0}%
8254   \bbl@elt{identification}{charset}{utf8}%
8255   \bbl@elt{identification}{version}{1.0}%
8256   \bbl@elt{identification}{date}{2022-05-16}%
8257   \bbl@elt{identification}{name.local}{nil}%
8258   \bbl@elt{identification}{name.english}{nil}%
8259   \bbl@elt{identification}{name.babel}{nil}%

```

```

8260 \bbl@elt{identification}{tag.bcp47}{und}%
8261 \bbl@elt{identification}{language.tag.bcp47}{und}%
8262 \bbl@elt{identification}{tag.opentype}{dflt}%
8263 \bbl@elt{identification}{script.name}{Latin}%
8264 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8265 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8266 \bbl@elt{identification}{level}{1}%
8267 \bbl@elt{identification}{encodings}{}%
8268 \bbl@elt{identification}{derivate}{no}}
8269 \@namedef{bbl@tbc@nil}{und}
8270 \@namedef{bbl@lbc@nil}{und}
8271 \@namedef{bbl@casing@nil}{und} % TODO
8272 \@namedef{bbl@lotf@nil}{dflt}
8273 \@namedef{bbl@elname@nil}{nil}
8274 \@namedef{bbl@lname@nil}{nil}
8275 \@namedef{bbl@esname@nil}{Latin}
8276 \@namedef{bbl@sname@nil}{Latin}
8277 \@namedef{bbl@sbc@nil}{Latn}
8278 \@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.

```

8279 \ldf@finish{nil}
8280 </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.

```

8281 <<Compute Julian day>> ≡
8282 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8283 \def\bbl@cs@gregleap#1{%
8284   (\bbl@fpmmod{#1}{4} == 0) &&
8285   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8286 \def\bbl@cs@jd#1#2#3{% year, month, day
8287   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8288     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8289     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
8290     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
8291 <</Compute Julian day>>

```

13.1. Islamic

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

```

8292 <*ca-islamic>
8293 \ExplSyntaxOn
8294 <@Compute Julian day@>
8295 % == islamic (default)
8296 % Not yet implemented
8297 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar.

```

8298 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8299   ((#3 + ceil(29.5 * (#2 - 1)) +
8300     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8301     1948439.5) - 1) }
8302 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8303 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8304 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8305 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}

```

```

8306 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8307 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8308   \edef\bbl@tempa{%
8309     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8310   \edef#5{%
8311     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8312   \edef#6{\fp_eval:n{
8313     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8314   \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).

```

8315 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8316 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8317 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8318 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8319 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8320 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8321 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8322 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8323 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8324 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8325 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8326 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8327 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8328 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8329 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8330 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8331 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8332 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8333 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8334 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8335 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8336 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8337 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8338 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8339 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8340 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8341 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8342 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8343 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8344 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8345 65401,65431,65460,65490,65520}
8346 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8347 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8348 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{0}}
8349 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8350   \ifnum#2>2014 \ifnum#2<2038
8351     \bbl@afterfi\expandafter\@gobble
8352   \fi\fi
8353   {\bbl@error{year-out-range}{2014-2038}{}}}%
8354   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8355     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8356   \count@\@ne
8357   \bbl@foreach\bbl@cs@umalqura@data{%
8358     \advance\count@\@ne
8359     \ifnum##1>\bbl@tempd\else
8360       \edef\bbl@tempe{\the\count@}%
8361       \edef\bbl@tempb{##1}%
8362       \fi}%
8363   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar

```

```

8364 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8365 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8366 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8367 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8368 \ExplSyntaxOff
8369 \bbl@add\bbl@precalendar{%
8370   \bbl@replace\bbl@ld@calendar{-civil}{}}%
8371   \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8372   \bbl@replace\bbl@ld@calendar{+}{}}%
8373   \bbl@replace\bbl@ld@calendar{-}{}}%
8374 </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`

```

8375 < *ca-hebrew>
8376 \newcount\bbl@cntcommon
8377 \def\bbl@remainder#1#2#3{%
8378   #3=#1\relax
8379   \divide #3 by #2\relax
8380   \multiply #3 by -#2\relax
8381   \advance #3 by #1\relax}%
8382 \newif\ifbbl@divisible
8383 \def\bbl@checkifdivisible#1#2{%
8384   {\countdef\tmp=0
8385     \bbl@remainder{#1}{#2}{\tmp}%
8386     \ifnum \tmp=0
8387       \global\bbl@divisibletrue
8388     \else
8389       \global\bbl@divisiblefalse
8390     \fi}}
8391 \newif\ifbbl@gregleap
8392 \def\bbl@ifgregleap#1{%
8393   \bbl@checkifdivisible{#1}{4}%
8394   \ifbbl@divisible
8395     \bbl@checkifdivisible{#1}{100}%
8396     \ifbbl@divisible
8397       \bbl@checkifdivisible{#1}{400}%
8398       \ifbbl@divisible
8399         \bbl@gregleaptrue
8400       \else
8401         \bbl@gregleapfalse
8402       \fi
8403     \else
8404       \bbl@gregleaptrue
8405     \fi
8406   \else
8407     \bbl@gregleapfalse
8408   \fi
8409   \ifbbl@gregleap}
8410 \def\bbl@gregdayspriormonths#1#2#3{%
8411   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8412     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8413   \bbl@ifgregleap{#2}%
8414   \ifnum #1 > 2
8415     \advance #3 by 1
8416   \fi
8417   \fi
8418   \global\bbl@cntcommon=#3}%
8419   #3=\bbl@cntcommon}
8420 \def\bbl@gregdaysprioryears#1#2{%

```



```

8421 {\countdef\tmpc=4
8422 \countdef\tmpb=2
8423 \tmpb=#1\relax
8424 \advance \tmpb by -1
8425 \tmpc=\tmpb
8426 \multiply \tmpc by 365
8427 #2=\tmpc
8428 \tmpc=\tmpb
8429 \divide \tmpc by 4
8430 \advance #2 by \tmpc
8431 \tmpc=\tmpb
8432 \divide \tmpc by 100
8433 \advance #2 by -\tmpc
8434 \tmpc=\tmpb
8435 \divide \tmpc by 400
8436 \advance #2 by \tmpc
8437 \global\bbl@cntcommon=#2\relax}%
8438 #2=\bbl@cntcommon}
8439 \def\bbl@absfromgreg#1#2#3#4{%
8440 {\countdef\tmpd=0
8441 #4=#1\relax
8442 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8443 \advance #4 by \tmpd
8444 \bbl@gregdaysprioryears{#3}{\tmpd}%
8445 \advance #4 by \tmpd
8446 \global\bbl@cntcommon=#4\relax}%
8447 #4=\bbl@cntcommon}
8448 \newif\ifbbl@hebrleap
8449 \def\bbl@checkleaphebryear#1{%
8450 {\countdef\tmpa=0
8451 \countdef\tmpb=1
8452 \tmpa=#1\relax
8453 \multiply \tmpa by 7
8454 \advance \tmpa by 1
8455 \bbl@remainder{\tmpa}{19}{\tmpb}%
8456 \ifnum \tmpb < 7
8457 \global\bbl@hebrleaptrue
8458 \else
8459 \global\bbl@hebrleapfalse
8460 \fi}}
8461 \def\bbl@hebrlapsedmonths#1#2{%
8462 {\countdef\tmpa=0
8463 \countdef\tmpb=1
8464 \countdef\tmpc=2
8465 \tmpa=#1\relax
8466 \advance \tmpa by -1
8467 #2=\tmpa
8468 \divide #2 by 19
8469 \multiply #2 by 235
8470 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8471 \tmpc=\tmpb
8472 \multiply \tmpb by 12
8473 \advance #2 by \tmpb
8474 \multiply \tmpc by 7
8475 \advance \tmpc by 1
8476 \divide \tmpc by 19
8477 \advance #2 by \tmpc
8478 \global\bbl@cntcommon=#2}%
8479 #2=\bbl@cntcommon}
8480 \def\bbl@hebrlapseddays#1#2{%
8481 {\countdef\tmpa=0
8482 \countdef\tmpb=1
8483 \countdef\tmpc=2

```

```

8484 \bbl@hebreleapsedmonths{#1}{#2}%
8485 \tmpa=#2\relax
8486 \multiply \tmpa by 13753
8487 \advance \tmpa by 5604
8488 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8489 \divide \tmpa by 25920
8490 \multiply #2 by 29
8491 \advance #2 by 1
8492 \advance #2 by \tmpa
8493 \bbl@remainder{#2}{7}{\tmpa}%
8494 \ifnum \tmpc < 19440
8495     \ifnum \tmpc < 9924
8496     \else
8497         \ifnum \tmpa=2
8498             \bbl@checkleaphebrewyear{#1}% of a common year
8499             \ifbbl@hebrleap
8500             \else
8501                 \advance #2 by 1
8502             \fi
8503         \fi
8504     \fi
8505     \ifnum \tmpc < 16789
8506     \else
8507         \ifnum \tmpa=1
8508             \advance #1 by -1
8509             \bbl@checkleaphebrewyear{#1}% at the end of leap year
8510             \ifbbl@hebrleap
8511                 \advance #2 by 1
8512             \fi
8513         \fi
8514     \fi
8515 \else
8516     \advance #2 by 1
8517 \fi
8518 \bbl@remainder{#2}{7}{\tmpa}%
8519 \ifnum \tmpa=0
8520     \advance #2 by 1
8521 \else
8522     \ifnum \tmpa=3
8523         \advance #2 by 1
8524     \else
8525         \ifnum \tmpa=5
8526             \advance #2 by 1
8527         \fi
8528     \fi
8529 \fi
8530 \global\bbl@cntcommon=#2\relax}%
8531 #2=\bbl@cntcommon}
8532 \def\bbl@daysinhebrewyear#1#2{%
8533 {\countdef\tmpe=12
8534 \bbl@hebreleapseddays{#1}{\tmpe}%
8535 \advance #1 by 1
8536 \bbl@hebreleapseddays{#1}{#2}%
8537 \advance #2 by -\tmpe
8538 \global\bbl@cntcommon=#2}%
8539 #2=\bbl@cntcommon}
8540 \def\bbl@hebrdayspriormonths#1#2#3{%
8541 {\countdef\tmpf= 14
8542 #3=\ifcase #1
8543     0 \or
8544     0 \or
8545     30 \or
8546     59 \or

```

```

8547         89 \or
8548         118 \or
8549         148 \or
8550         148 \or
8551         177 \or
8552         207 \or
8553         236 \or
8554         266 \or
8555         295 \or
8556         325 \or
8557         400
8558     \fi
8559     \bbl@checkleaphebyear{#2}%
8560     \ifbbl@hebrleap
8561         \ifnum #1 > 6
8562             \advance #3 by 30
8563         \fi
8564     \fi
8565     \bbl@daysinhebyear{#2}{\tmpf}%
8566     \ifnum #1 > 3
8567         \ifnum \tmpf=353
8568             \advance #3 by -1
8569         \fi
8570         \ifnum \tmpf=383
8571             \advance #3 by -1
8572         \fi
8573     \fi
8574     \ifnum #1 > 2
8575         \ifnum \tmpf=355
8576             \advance #3 by 1
8577         \fi
8578         \ifnum \tmpf=385
8579             \advance #3 by 1
8580         \fi
8581     \fi
8582     \global\bbl@cntcommon=#3\relax}%
8583     #3=\bbl@cntcommon}
8584 \def\bbl@absfromhebr#1#2#3#4{%
8585     {#4=#1\relax
8586     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8587     \advance #4 by #1\relax
8588     \bbl@hebrrelapseddays{#3}{#1}%
8589     \advance #4 by #1\relax
8590     \advance #4 by -1373429
8591     \global\bbl@cntcommon=#4\relax}%
8592     #4=\bbl@cntcommon}
8593 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8594     {\countdef\tmpx= 17
8595     \countdef\tmpy= 18
8596     \countdef\tmpz= 19
8597     #6=#3\relax
8598     \global\advance #6 by 3761
8599     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8600     \tmpz=1 \tmpy=1
8601     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8602     \ifnum \tmpx > #4\relax
8603         \global\advance #6 by -1
8604         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8605     \fi
8606     \advance #4 by -\tmpx
8607     \advance #4 by 1
8608     #5=#4\relax
8609     \divide #5 by 30

```

```

8610 \loop
8611 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8612 \ifnum \tmpx < #4\relax
8613 \advance #5 by 1
8614 \tmpy=\tmpx
8615 \repeat
8616 \global\advance #5 by -1
8617 \global\advance #4 by -\tmpy}}
8618 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8619 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8620 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8621 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8622 \bbl@hebrfromgreg
8623 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8624 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8625 \edef#4{\the\bbl@hebyear}%
8626 \edef#5{\the\bbl@hebrmonth}%
8627 \edef#6{\the\bbl@hebrday}}
8628 </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).

```

8629 <*ca-persian>
8630 \ExplSyntaxOn
8631 <@Compute Julian day@>
8632 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8633 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8634 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8635 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8636 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8637 \bbl@afterfi\expandafter\@gobble
8638 \fi\fi
8639 {\bbl@error{year-out-range}{2013-2050}{}}}%
8640 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8641 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8642 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8643 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8644 \ifnum\bbl@tempc<\bbl@tempb
8645 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8646 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8647 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8648 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8649 \fi
8650 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8651 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8652 \edef#5{\fp_eval:n{% set Jalali month
8653 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8654 \edef#6{\fp_eval:n{% set Jalali day
8655 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}%
8656 \ExplSyntaxOff
8657 </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.

```

8658 <*ca-coptic>

```

```

8659 \ExplSyntaxOn
8660 <@Compute Julian day@>
8661 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8662   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8663   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8664   \edef#4{\fp_eval:n{%
8665     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8666   \edef\bbl@tempc{\fp_eval:n{%
8667     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8668   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8669   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8670 \ExplSyntaxOff
8671 </ca-coptic>
8672 <*ca-ethiopic>
8673 \ExplSyntaxOn
8674 <@Compute Julian day@>
8675 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8676   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8677   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8678   \edef#4{\fp_eval:n{%
8679     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8680   \edef\bbl@tempc{\fp_eval:n{%
8681     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8682   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8683   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8684 \ExplSyntaxOff
8685 </ca-ethiopic>

```

13.5. Buddhist

That's very simple.

```

8686 <*ca-buddhist>
8687 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8688   \edef#4{\number\numexpr#1+543\relax}%
8689   \edef#5{#2}%
8690   \edef#6{#3}}
8691 </ca-buddhist>
8692 %
8693 % \subsection{Chinese}
8694 %
8695 % Brute force, with the Julian day of first day of each month. The
8696 % table has been computed with the help of \textsf{python-lunardate} by
8697 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8698 % is 2015-2044.
8699 %
8700 % \begin{macrocode}
8701 <*ca-chinese>
8702 \ExplSyntaxOn
8703 <@Compute Julian day@>
8704 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8705   \edef\bbl@tempd{\fp_eval:n{%
8706     \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8707   \count@ \z@
8708   \@tempcnta=2015
8709   \bbl@foreach\bbl@cs@chinese@data{%
8710     \ifnum##1>\bbl@tempd\else
8711       \advance\count@\@ne
8712       \ifnum\count@>12
8713         \count@\@ne
8714         \advance\@tempcnta\@ne\fi
8715       \bbl@xin@{,##1,}{, \bbl@cs@chinese@leap,}%
8716       \ifin@
8717         \advance\count@\m@ne

```

```

8718 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8719 \else
8720 \edef\bbl@tempe{\the\count@}%
8721 \fi
8722 \edef\bbl@tempb{##1}%
8723 \fi}%
8724 \edef#4{\the\@tempcnta}%
8725 \edef#5{\bbl@tempe}%
8726 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8727 \def\bbl@cs@chinese@leap{%
8728 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8729 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8730 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8731 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8732 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8733 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8734 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8735 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8736 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8737 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8738 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8739 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8740 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8741 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8742 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8743 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8744 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8745 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8746 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8747 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8748 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8749 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8750 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8751 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8752 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8753 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8754 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8755 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8756 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8757 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8758 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8759 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8760 10896,10926,10956,10986,11015,11045,11074,11103}
8761 \ExplSyntaxOff
8762 </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 \LaTeX sees, we need to set some category codes just to be able to change the definition of \input .

```
8763 <*\bplain | blplain>
8764 \catcode`\{=1 % left brace is begin-group character
8765 \catcode`\}=2 % right brace is end-group character
8766 \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).

```
8767 \openin 0 hyphen.cfg
8768 \ifeof0
8769 \else
8770 \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.

```
8771 \def\input #1 {%
8772 \let\input\input
8773 \a hyphen.cfg
8774 \let\input\undefined
8775 }
8776 \fi
```

```
8777 </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`.

```
8778 <bplain>\a plain.tex
8779 <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.

```
8780 <bplain>\def\fmtname{babel-plain}
8781 <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 $\text{\babeloptionstrings}$ and \babeloptionmath are provided, which can be defined before loading `babel`. \BabelModifiers can be set too (but not sure it works).

```
8782 <<*\Emulate LaTeX>> \equiv
8783 \def\@empty{}
8784 \def\loadlocalcfg#1{%
8785 \openin0#1.cfg
8786 \ifeof0
8787 \closein0
8788 \else
8789 \closein0
8790 {\immediate\writel6{*****}%
8791 \immediate\writel6{* Local config file #1.cfg used}%
8792 \immediate\writel6{*}%
8793 }
8794 \input #1.cfg\relax
8795 \fi
8796 \@endofldef}
```

14.3. General tools

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

```
8797 \long\def\@firstofone#1{#1}
8798 \long\def\@firstoftwo#1#2{#1}
8799 \long\def\@secondoftwo#1#2{#2}
8800 \def\@nnil{\@nil}
8801 \def\@gobbletwo#1#2{}
8802 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8803 \def\@star@or@long#1{%
8804   \@ifstar
8805   {\let\l@ngrel@x\relax#1}%
8806   {\let\l@ngrel@x\long#1}}
8807 \let\l@ngrel@x\relax
8808 \def\@car#1#2\@nil{#1}
8809 \def\@cdr#1#2\@nil{#2}
8810 \let\@typeset@protect\relax
8811 \let\protected@edef\edef
8812 \long\def\@gobble#1{}
8813 \edef\@backslashchar{\expandafter\@gobble\string\}
8814 \def\strip@prefix#1>{}
8815 \def\g@addto@macro#1#2{{%
8816   \toks@\expandafter{#1#2}%
8817   \xdef#1{\the\toks@}}}
8818 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8819 \def\@nameuse#1{\csname #1\endcsname}
8820 \def\@ifundefined#1{%
8821   \expandafter\ifx\csname#1\endcsname\relax
8822     \expandafter\@firstoftwo
8823   \else
8824     \expandafter\@secondoftwo
8825   \fi}
8826 \def\@expandtwoargs#1#2#3{%
8827   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8828 \def\zap@space#1 #2{%
8829   #1%
8830   \ifx#2\@empty\else\expandafter\zap@space\fi
8831   #2}
8832 \let\bbl@trace\@gobble
8833 \def\bbl@error#1{% Implicit #2#3#4
8834   \begingroup
8835     \catcode`\=0 \catcode`\==12 \catcode`\`=12
8836     \catcode`\^M=5 \catcode`\%=14
8837     \input errbabel.def
8838   \endgroup
8839   \bbl@error{#1}}
8840 \def\bbl@warning#1{%
8841   \begingroup
8842     \newlinechar=`^^J
8843     \def\{`^^J(babel) }%
8844     \message{\{`^^J}%
8845   \endgroup}
8846 \let\bbl@infowarn\bbl@warning
8847 \def\bbl@info#1{%
8848   \begingroup
8849     \newlinechar=`^^J
8850     \def\{`^^J}%
8851     \wlog{#1}%
8852   \endgroup}
```

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

```
8853 \ifx\@preamblecmds\undefined
```



```

8854 \def\@preamblecmds{}
8855 \fi
8856 \def\@onlypreamble#1{%
8857 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8858 \@preamblecmds\do#1}}
8859 \@onlypreamble\@onlypreamble

```

Mimic L^AT_EX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```

8860 \def\begindocument{%
8861 \@begindocumenthook
8862 \global\let\@begindocumenthook\@undefined
8863 \def\do##1{\global\let##1\@undefined}%
8864 \@preamblecmds
8865 \global\let\do\noexpand}

8866 \ifx\@begindocumenthook\@undefined
8867 \def\@begindocumenthook{}
8868 \fi
8869 \@onlypreamble\@begindocumenthook
8870 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8871 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
8872 \@onlypreamble\AtEndOfPackage
8873 \def\@endoflfd{}
8874 \@onlypreamble\@endoflfd
8875 \let\bbl@afterlang\@empty
8876 \chardef\bbl@opt@hyphenmap\z@

```

L^AT_EX 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.

```

8877 \catcode`\&=\z@
8878 \ifx&\if@files\@undefined
8879 \expandafter\let\csname if@files\expandafter\endcsname
8880 \csname iffalse\endcsname
8881 \fi
8882 \catcode`\&=4

```

Mimic L^AT_EX's commands to define control sequences.

```

8883 \def\newcommand{\@star@or@long\new@command}
8884 \def\new@command#1{%
8885 \@testopt{\@newcommand#1}0}
8886 \def\@newcommand#1[#2]{%
8887 \@ifnextchar [{\@xargdef#1[#2]}%
8888 {\@argdef#1[#2]}}
8889 \long\def\@argdef#1[#2]#3{%
8890 \@yargdef#1\@ne{#2}{#3}}
8891 \long\def\@xargdef#1[#2][#3]#4{%
8892 \expandafter\def\expandafter#1\expandafter{%
8893 \expandafter\@protected@testopt\expandafter #1%
8894 \csname\string#1\expandafter\endcsname{#3}}}%
8895 \expandafter\@yargdef \csname\string#1\endcsname
8896 \tw@{#2}{#4}}
8897 \long\def\@yargdef#1#2#3{%
8898 \@tempcnta#3\relax
8899 \advance \@tempcnta \@ne
8900 \let\@hash@\relax
8901 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8902 \@tempcntb #2%
8903 \@whilenum \@tempcntb < \@tempcnta
8904 \do{%
8905 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%

```

```

8906 \advance\@tempcntb \@ne}%
8907 \let\@hash@##%
8908 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8909 \def\providecommand{\@star@or@long\provide@command}
8910 \def\provide@command#1{%
8911 \begingroup
8912 \escapechar\m@ne\xdef\@gtempa{\string#1}%
8913 \endgroup
8914 \expandafter\ifundefined\@gtempa
8915 {\def\reserved@a{\new@command#1}}%
8916 {\let\reserved@a\relax
8917 \def\reserved@a{\new@command\reserved@a}}%
8918 \reserved@a}%

8919 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8920 \def\declare@robustcommand#1{%
8921 \edef\reserved@a{\string#1}%
8922 \def\reserved@b{#1}%
8923 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8924 \edef#1{%
8925 \ifx\reserved@a\reserved@b
8926 \noexpand\x@protect
8927 \noexpand#1%
8928 \fi
8929 \noexpand\protect
8930 \expandafter\noexpand\csname
8931 \expandafter\@gobble\string#1 \endcsname
8932 }%
8933 \expandafter\new@command\csname
8934 \expandafter\@gobble\string#1 \endcsname
8935 }
8936 \def\x@protect#1{%
8937 \ifx\protect\@typeset@protect\else
8938 \x@protect#1%
8939 \fi
8940 }
8941 \catcode`\&=\z@ % Trick to hide conditionals
8942 \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`.

```

8943 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8944 \catcode`\&=4
8945 \ifx\in@\@undefined
8946 \def\in@#1#2{%
8947 \def\in@##1#1##2##3\in@{%
8948 \ifx\in@##2\in@false\else\in@true\fi}%
8949 \in@##2#1\in@\in@}
8950 \else
8951 \let\bbl@tempa\@empty
8952 \fi
8953 \bbl@tempa

```

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

```

8954 \def\@ifpackagewith#1#2#3#4{#3}

```

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

```

8955 \def\@ifl@aded#1#2#3#4{}

```

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

```
8956 \ifx\@tempcnta\@undefined
8957   \csname newcount\endcsname\@tempcnta\relax
8958 \fi
8959 \ifx\@tempcntb\@undefined
8960   \csname newcount\endcsname\@tempcntb\relax
8961 \fi
```

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

```
8962 \ifx\bye\@undefined
8963   \advance\count10 by -2\relax
8964 \fi
8965 \ifx\@ifnextchar\@undefined
8966   \def\@ifnextchar#1#2#3{%
8967     \let\reserved@d=#1%
8968     \def\reserved@a{#2}\def\reserved@b{#3}%
8969     \futurelet\@let@token\@ifnch}
8970 \def\@ifnch{%
8971   \ifx\@let@token\@sptoken
8972     \let\reserved@c\@xifnch
8973   \else
8974     \ifx\@let@token\reserved@d
8975       \let\reserved@c\reserved@a
8976     \else
8977       \let\reserved@c\reserved@b
8978     \fi
8979   \fi
8980   \reserved@c}
8981 \def\{\let\@sptoken= } \: % this makes \@sptoken a space token
8982 \def\{\@xifnch} \expandafter\def\{\futurelet\@let@token\@ifnch}
8983 \fi
8984 \def\@testopt#1#2{%
8985   \@ifnextchar[#{1}{#1[#2]}}
8986 \def\@protected@testopt#1{%
8987   \ifx\protect\@typeset@protect
8988     \expandafter\@testopt
8989   \else
8990     \@x@protect#1%
8991   \fi}
8992 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8993   #2\relax}\fi}
8994 \long\def\@iwhilenum#1{\ifnum #1\relax\expandafter\@iwhilenum
8995   \else\expandafter\@gobble\fi{#1}}
```

14.4. Encoding related macros

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

```
8996 \def\DeclareTextCommand{%
8997   \@dec@text@cmd\providecommand
8998 }
8999 \def\ProvideTextCommand{%
9000   \@dec@text@cmd\providecommand
9001 }
9002 \def\DeclareTextSymbol#1#2#3{%
9003   \@dec@text@cmd\chardef#1{#2}#3\relax
9004 }
9005 \def\@dec@text@cmd#1#2#3{%
9006   \expandafter\def\expandafter#2%
9007     \expandafter{%
```

```

9008         \csname#3-cmd\expandafter\endcsname
9009         \expandafter#2%
9010         \csname#3\string#2\endcsname
9011     }%
9012 % \let\@ifdefinable\@rc@ifdefinable
9013 \expandafter#1\csname#3\string#2\endcsname
9014 }
9015 \def\@current@cmd#1{%
9016 \ifx\protect\@typeset@protect\else
9017     \noexpand#1\expandafter\@gobble
9018 \fi
9019 }
9020 \def\@changed@cmd#1#2{%
9021 \ifx\protect\@typeset@protect
9022     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9023     \expandafter\ifx\csname ?\string#1\endcsname\relax
9024     \expandafter\def\csname ?\string#1\endcsname{%
9025         \@changed@x@err{#1}%
9026     }%
9027 \fi
9028 \global\expandafter\let
9029     \csname\cf@encoding \string#1\expandafter\endcsname
9030     \csname ?\string#1\endcsname
9031 \fi
9032 \csname\cf@encoding\string#1%
9033 \expandafter\endcsname
9034 \else
9035     \noexpand#1%
9036 \fi
9037 }
9038 \def\@changed@x@err#1{%
9039 \errhelp{Your command will be ignored, type <return> to proceed}%
9040 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9041 \def\DeclareTextCommandDefault#1{%
9042 \DeclareTextCommand#1?%
9043 }
9044 \def\ProvideTextCommandDefault#1{%
9045 \ProvideTextCommand#1?%
9046 }
9047 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9048 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9049 \def\DeclareTextAccent#1#2#3{%
9050 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9051 }
9052 \def\DeclareTextCompositeCommand#1#2#3#4{%
9053 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9054 \edef\reserved@b{\string##1}%
9055 \edef\reserved@c{%
9056     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9057 \ifx\reserved@b\reserved@c
9058     \expandafter\expandafter\expandafter\ifx
9059         \expandafter\@car\reserved@a\relax\relax\@nil
9060         \@text@composite
9061 \else
9062     \edef\reserved@b##1{%
9063         \def\expandafter\noexpand
9064             \csname#2\string#1\endcsname###1{%
9065             \noexpand\@text@composite
9066                 \expandafter\noexpand\csname#2\string#1\endcsname
9067                 ###1\noexpand\@empty\noexpand\@text@composite
9068                 {##1}%
9069             }%
9070     }%

```

```

9071 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9072 \fi
9073 \expandafter\def\csname\expandafter\string\csname
9074 #2\endcsname\string#1-\string#3\endcsname{#4}
9075 \else
9076 \errhelp{Your command will be ignored, type <return> to proceed}%
9077 \errmessage{\string\DeclareTextCompositeCommand\space used on
9078 inappropriate command \protect#1}
9079 \fi
9080 }
9081 \def\@text@composite#1#2#3\@text@composite{%
9082 \expandafter\@text@composite@x
9083 \csname\string#1-\string#2\endcsname
9084 }
9085 \def\@text@composite@x#1#2{%
9086 \ifx#1\relax
9087 #2%
9088 \else
9089 #1%
9090 \fi
9091 }
9092 %
9093 \def\@strip@args#1:#2-#3\@strip@args{#2}
9094 \def\DeclareTextComposite#1#2#3#4{%
9095 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9096 \bgroup
9097 \lccode`\@=#4%
9098 \lowercase{%
9099 \egroup
9100 \reserved@a \@%
9101 }%
9102 }
9103 %
9104 \def\UseTextSymbol#1#2{#2}
9105 \def\UseTextAccent#1#2#3{}
9106 \def\@use@text@encoding#1{}
9107 \def\DeclareTextSymbolDefault#1#2{%
9108 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9109 }
9110 \def\DeclareTextAccentDefault#1#2{%
9111 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9112 }
9113 \def\cf@encoding{OT1}

```

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

```

9114 \DeclareTextAccent{"}{OT1}{127}
9115 \DeclareTextAccent{'}{OT1}{19}
9116 \DeclareTextAccent{^}{OT1}{94}
9117 \DeclareTextAccent{\`}{OT1}{18}
9118 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9119 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9120 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
9121 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9122 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
9123 \DeclareTextSymbol{\i}{OT1}{16}
9124 \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`.

```

9125 \ifx\scriptsize\undefined
9126 \let\scriptsize\sevenrm

```

```

9127 \fi

And a few more “dummy” definitions.

9128 \def\language{english}%
9129 \let\bbl@opt@shorthands\@nnil
9130 \def\bbl@ifshorthand#1#2#3{#2}%
9131 \let\bbl@language@opts\@empty
9132 \let\bbl@ensureinfo\@gobble
9133 \let\bbl@provide@locale\relax
9134 \ifx\babeloptionstrings\undefined
9135   \let\bbl@opt@strings\@nnil
9136 \else
9137   \let\bbl@opt@strings\babeloptionstrings
9138 \fi
9139 \def\BabelStringsDefault{generic}
9140 \def\bbl@tempa{normal}
9141 \ifx\babeloptionmath\bbl@tempa
9142   \def\bbl@mathnormal{\noexpand\textormath}
9143 \fi
9144 \def\AfterBabelLanguage#1#2{}
9145 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9146 \let\bbl@afterlang\relax
9147 \def\bbl@opt@safe{BR}
9148 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
9149 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9150 \expandafter\newif\csname ifbbl@single\endcsname
9151 \chardef\bbl@bidimode\z@
9152 <</Emulate LaTeX>>

A proxy file:

9153 <*\plain>
9154 \input babel.def
9155 </\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 T_EX*, *TUGboat* 9 (1988) #1, pp. 70–72.
- [11] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, pp. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, pp. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).