

Babel

Code

Version 25.11.93475
2025/07/22

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

LuaT_EX

pdfT_EX

XeT_EX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	A few core definitions	8
3.2	LaTeX: babel.sty (start)	8
3.3	base	9
3.4	key=value options and other general option	10
3.5	Post-process some options	11
3.6	Plain: babel.def (start)	13
4	babel.sty and babel.def (common)	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	More on selection	23
4.4	Short tags	25
4.5	Compatibility with language.def	25
4.6	Hooks	26
4.7	Setting up language files	27
4.8	Shorthands	29
4.9	Language attributes	38
4.10	Support for saving and redefining macros	39
4.11	French spacing	40
4.12	Hyphens	41
4.13	Multiencoding strings	43
4.14	Tailor captions	48
4.15	Making glyphs available	49
4.15.1	Quotation marks	49
4.15.2	Letters	50
4.15.3	Shorthands for quotation marks	51
4.15.4	Umlauts and tremas	52
4.16	Layout	53
4.17	Load engine specific macros	54
4.18	Creating and modifying languages	54
4.19	Main loop in ‘provide’	61
4.20	Processing keys in ini	66
4.21	French spacing (again)	71
4.22	Handle language system	73
4.23	Numerals	73
4.24	Casing	74
4.25	Getting info	75
4.26	BCP 47 related commands	76
5	Adjusting the Babel behavior	77
5.1	Cross referencing macros	79
5.2	Layout	82
5.3	Marks	84
5.4	Other packages	85
5.4.1	ifthen	85
5.4.2	varioref	85
5.4.3	hhline	86
5.5	Encoding and fonts	86
5.6	Basic bidi support	88
5.7	Local Language Configuration	91
5.8	Language options	92

6	The kernel of Babel	96
7	Error messages	96
8	Loading hyphenation patterns	99
9	luatex + xetex: common stuff	103
10	Hooks for XeTeX and LuaTeX	107
10.1	XeTeX	107
10.2	Support for interchar	108
10.3	Layout	110
10.4	8-bit TeX	112
10.5	LuaTeX	113
10.6	Southeast Asian scripts	120
10.7	CJK line breaking	121
10.8	Arabic justification	123
10.9	Common stuff	127
10.10	Automatic fonts and ids switching	127
10.11	Bidi	134
10.12	Layout	137
10.13	Lua: transforms	146
10.14	Lua: Auto bidi with basic and basic-r	156
11	Data for CJK	168
12	The ‘nil’ language	168
13	Calendars	169
13.1	Islamic	169
13.2	Hebrew	171
13.3	Persian	175
13.4	Coptic and Ethiopic	175
13.5	Buddhist	176
14	Support for Plain T_EX (plain.def)	177
14.1	Not renaming hyphen.tex	177
14.2	Emulating some L ^A T _E X features	178
14.3	General tools	178
14.4	Encoding related macros	182
15	Acknowledgements	185

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

1. Identification and loading of required files

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files.

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with `<@name@>` at the appropriate places in the source code and defined with either `<<name=value>>`, or with a series of lines between `<<*name>>` and `<</name>>`. The latter is cumulative (e.g., with *More package options*). That brings a little bit of literate programming. The guards `<-name>` and `<+name>` have been redefined, too. See babel.ins for further details.

2. locale directory

A required component of babel is a set of ini files with basic definitions for about 300 languages. They are distributed as a separate zip file, not packed as dtx. Many of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (e.g., there are no geographic areas in Spanish). Not all include LICR variants.

babel-*.ini files contain the actual data; babel-*.tex files are basically proxies to the corresponding ini files.

See [Keys in ini files](#) in the the babel site.

3. Tools

```
1 <<version=25.11.93475>>
2 <<date=2025/07/22>>
```

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

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

```

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

```

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

```

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

```

\bbl@afterelse

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

For each element in the comma separated `<key>=<value>` list, execute `<code>` with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the `<key>` alone, it passes `\@empty` (i.e., the macro thus named, not an empty argument, which is what you get with `<key>=` and no value).

```

81 \def\bbl@forkv#1#2{%
82   \def\bbl@kvcmd##1##2##3{#2}%
83   \bbl@kvnext#1,\@nil,}
84 \def\bbl@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86   \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
87   \expandafter\bbl@kvnext
88   \fi}
89 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
90   \bbl@trim\def\bbl@forkv@a{#1}%
91   \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}

```

A *for* loop. Each item (trimmed) is #1. It cannot be nested (it’s doable, but we don’t need it).

```

92 \def\bbl@vforeach#1#2{%
93   \def\bbl@forcmd##1{#2}%
94   \bbl@fornext#1,\@nil,}
95 \def\bbl@fornext#1,{%
96   \ifx\@nil#1\relax\else
97   \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
98   \expandafter\bbl@fornext
99   \fi}
100 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

Some code should be executed once. The first argument is a flag.

```

101 \global\let\bbl@done\@empty

```

```

102 \def\bbl@once#1#2{%
103   \bbl@xin@{,#1,}{,\bbl@done,}%
104   \ifin@ \else
105     #2%
106   \xdef\bbl@done{\bbl@done,#1,}%
107   \fi}
108 %   \end{macrode}
109 %
110 % \macro{\bbl@replace}
111 %
112 % Returns implicitly |\toks@| with the modified string.
113 %
114 %   \begin{macrocode}
115 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
116   \toks@{ }%
117   \def\bbl@replace@aux##1#2##2#2{%
118     \ifx\bbl@nil##2%
119       \toks@\expandafter{\the\toks@##1}%
120     \else
121       \toks@\expandafter{\the\toks@##1#3}%
122       \bbl@afterfi
123       \bbl@replace@aux##2#2%
124     \fi}%
125   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
126   \edef#1{\the\toks@}}

```

An extension to the previous macro. It takes into account the parameters, and it is string based (i.e., if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in \bbl@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure checking the replacement is really necessary or just paranoia).

```

127 \ifx\detokenize@undefined\else % Unused macros if old Plain TeX
128   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax}%
129   \def\bbl@tempa{#1}%
130   \def\bbl@tempb{#2}%
131   \def\bbl@tempe{#3}}
132 \def\bbl@sreplace#1#2#3{%
133   \begingroup
134     \expandafter\bbl@parsedef\meaning#1\relax
135     \def\bbl@tempc{#2}%
136     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
137     \def\bbl@tempd{#3}%
138     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
139     \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
140     \ifin@
141       \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
142       \def\bbl@tempc{% Expanded an executed below as 'uplevel'
143         \\makeatletter % "internal" macros with @ are assumed
144         \\scantokens{%
145           \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}%
146           \noexpand\noexpand}%
147         \catcode64=\the\catcode64\relax}% Restore @
148     \else
149       \let\bbl@tempc@empty % Not \relax
150     \fi
151     \bbl@exp{% For the 'uplevel' assignments
152   \endgroup
153   \bbl@tempc}} % empty or expand to set #1 with changes
154 \fi

```

Two further tools. \bbl@ifsamestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbl@engine takes the following values: 0 is pdf_{La}TeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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


```

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

```

3.1. A few core definitions

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

```

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

```

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

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

```

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

```

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

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

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

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

```

223 <*package>
224 \NeedsTeXFormat{LaTeX2e}
225 \ProvidesPackage{babel}%
226 [<@date@> v<@version@>
227   The multilingual framework for LuaLaTeX, pdfLaTeX and XeLaTeX]

```

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

```

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

```

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

```

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

```

Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user.

But first, include here the *Basic macros* defined above.

```

265 <@Basic macros@>
266 \ifpackagewith{babel}{silent}
267   {\let\bbl@info\@gobble
268    \let\bbl@infowarn\@gobble
269    \let\bbl@warning\@gobble}
270 {}
271 %
272 \def\AfterBabelLanguage#1{%
273   \global\expandafter\bbl@add\csname#1.ldf-h@k\endcsname}%

```

If the format created a list of loaded languages (in \bbl@languages), get the name of the 0-th to show the actual language used. Also available with base, because it just shows info.

```

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

```

3.3. base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that \LaTeX forgets about the first loading. After a subset of `babel.def` has been loaded (the old `switch.def`) and `\AfterBabelLanguage` defined, it exits.

Now the base option. With it we can define (and load, with luatex) hyphenation patterns, even if we are not interested in the rest of babel.

```

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

```

3.4. key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to \BabelModifiers at \bbl@load@language; when no modifiers have been given, the former is \relax.

```

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

```

The next option tells babel to leave shorthand characters active at the end of processing the package. This is *not* the default as it can cause problems with other packages, but for those who want

to use the shorthand characters in the preamble of their documents this can help.

```

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

```

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax $\langle key \rangle = \langle value \rangle$, the second one loads the requested languages, except the main one if set with the key `main`, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```

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

```

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

```

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

```

Now the option list is processed, taking into account only currently declared options (including those declared with a `=`), and $\langle key \rangle = \langle value \rangle$ options (the former take precedence). Unrecognized options are saved in `\bbl@language@opts`, because they are language options.

```

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

```

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

```

380 \ProcessOptions*

```

3.5. Post-process some options

```

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

```

390 \fi

If there is no shorthands=*chars*, the original babel macros are left untouched, but if there is, these macros are wrapped (in babel.def) to define only those given.

A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=...

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

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

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

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

```
412 \edef\bbl@opt@shorthands{%
413   \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

The following is ignored with shorthands=off, since it is intended to take some additional actions for certain chars.

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

With headfoot=lang we can set the language used in heads/feet. For example, in babel/3796 just add headfoot=english. It misuses \resetactivechars, but seems to work.

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

For the option safe we use a different approach – \bbl@opt@safe says which macros are redefined (B for bibs and R for refs). By default, both are currently set, but in a future release it will be set to none.

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

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

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

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

```

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

```

3.6. Plain: babel.def (start)

Because of the way docstrip works, we need to insert some code for Plain here. However, the tools provided by the babel installer for literate programming makes this section a short interlude, because the actual code is below, tagged as *Emulate LaTeX*.

First, exit immediately if previously loaded.

```

448 \ifx\ldf@quit\undefined\else
449 \endinput\fi % Same line!
450 \ifx\ldf@quit\defined\else
451 \ifx\ldf@quit\defined\else
452 \ifx\ldf@quit\defined\else
453 \ifx\ldf@quit\defined\else
454 \ifx\ldf@quit\defined\else
455 \ifx\ldf@quit\defined\else
456 \ifx\ldf@quit\defined\else
457 \ifx\ldf@quit\defined\else

```

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

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

```

458 \ifx\ldf@quit\defined\else
459 \ifx\ldf@quit\defined\else
460 \ifx\ldf@quit\defined\else
461 \ifx\ldf@quit\defined\else

```

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

```

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

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error.

The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s an attempt to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility

(perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `\l` is encapsulated, so that its case does not change.

```

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

```

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP 47 code.

We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with `\bbl@bcpcase`, casing is the correct one, so that `sr-latn-ba` becomes `fr-Latn-BA`. Note #4 may contain some `\@empty`’s, but they are eventually removed.

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

```

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

```

```

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

```

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

```

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

```

4.1. Selecting the language

\selectlanguage It checks whether the language is already defined before it performs its actual task, which is to update `\language` and activate language-specific definitions.

```

543 \let\bbl@select@type\z@
544 \edef\selectlanguage{%
545 \noexpand\protect
546 \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

547 \ifx\@undefined\protect\let\protect\relax\fi

```

The following definition is preserved for backwards compatibility (e.g., *arabi*, *koma*). It is related to a trick for 2.09, now discarded.

```

548 \let\xstring\string

```

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

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

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

```

549 \def\bbl@language@stack{}

```

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

\bbl@push@language

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

```

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

```

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

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

```

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

```

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

```

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

```

Once the name of the previous language is retrieved from the stack, it is fed to \bbl@set@language to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of \localeid. This means \l@... will be reserved for hyphenation patterns (so that two locales can share the same rules).

```

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

```

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

```

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

```

```

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

```

\bbl@set@language The macro \bbl@set@language takes care of switching the language environment *and* of writing entries on the auxiliary files. For historical reasons, language names can be either language of \language. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in \language are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining \BabelContentsFiles, but make sure they are loaded inside a group (as aux, toc, lof, and lot do) or the last language of the document will remain active afterwards.

We also write a command to change the current language in the auxiliary files.

\bbl@savelastskip is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from hyperref, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in luatex, is to avoid the \write altogether when not needed).

```

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

```

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

```

```

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

```

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

```

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

```

738 \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`.

```

739 \expandafter\def\csname otherlanguage*\endcsname{%
740   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
741 \def\bbl@otherlanguage@s[#1]#2{%
742   \def\bbl@selectorname{other*}%
743   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
744   \def\bbl@select@opts{#1}%
745   \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”.

```

746 \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`.

```

747 \providecommand\bbl@beforeforeign{}
748 \edef\foreignlanguage{%
749   \noexpand\protect
750   \expandafter\noexpand\csname foreignlanguage \endcsname}
751 \expandafter\def\csname foreignlanguage \endcsname{%
752   \@ifstar\bbl@foreign@s\bbl@foreign@x}
753 \providecommand\bbl@foreign@x[3][]{%
754   \begingroup
755     \def\bbl@selectorname{foreign}%
756     \def\bbl@select@opts{#1}%
757     \let\BabelText\@firstofone
758     \bbl@beforeforeign
759     \foreign@language{#2}%
760     \bbl@usehooks{foreign}{}%
761     \BabelText{#3}% Now in horizontal mode!
762   \endgroup}
763 \def\bbl@foreign@s#1#2{%
764   \begingroup
765     {\par}%
766     \def\bbl@selectorname{foreign*}%
767     \let\bbl@select@opts\@empty
768     \let\BabelText\@firstofone
769     \foreign@language{#1}%
770     \bbl@usehooks{foreign*}{}%
771     \bbl@dirparastext
772     \BabelText{#2}% Still in vertical mode!
773     {\par}%
774   \endgroup}
775 \providecommand\BabelWrapText[1]{%
776   \def\bbl@tempa{\def\BabelText####1}%
777   \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.

```

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

```

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

```

791 \def\IfBabelSelectorTF#1{%
792   \bbl@xin{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
793   \ifin@
794     \expandafter\@firstoftwo
795   \else
796     \expandafter\@secondoftwo
797   \fi}

```

\bbl@patterns This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.

It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

```

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

```

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

```

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

```

841 \def\providehyphenmins#1#2{%
842   \expandafter\ifx\cename #1hyphenmins\endcename\relax
843   \@namedef{#1hyphenmins}{#2}%
844   \fi}

```

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

```

845 \def\set@hyphenmins#1#2{%
846   \lefthyphenmin#1\relax
847   \righthyphenmin#2\relax}

```

\ProvidesLanguage The identification code for each file is something that was introduced in \LaTeX 2_ϵ . 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.

```

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

```

864 \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.

```

865 \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’:

```
866 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}}
867 \let\uselocale\setlocale
868 \let\locale\setlocale
869 \let\selectlocale\setlocale
870 \let\textlocale\setlocale
871 \let\textlanguage\setlocale
872 \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 $\text{\LaTeX 2}\epsilon$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

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

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

903 \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.

```

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

```

```

960      \expandafter\bbl@tempa
961      \fi}%
962      \bbl@tempa#1\@empty}
963 \def\bbl@captionslist{%
964   \prefacename\refname\abstractname\bibname\chaptername\appendixname
965   \contentsname\listfigurename\listtablename\indexname\figurename
966   \tablename\partname\enclname\ccname\headtoname\pagename\seename
967   \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.

```

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

```

983 \bbl@trace{Compatibility with language.def}
984 \ifx\directlua\@undefined\else
985   \ifx\bbl@luapatterns\@undefined
986     \input luababel.def
987   \fi
988 \fi
989 \ifx\bbl@languages\@undefined
990   \ifx\directlua\@undefined
991     \openin1 = language.def
992     \ifeof1
993       \closein1
994       \message{I couldn't find the file language.def}
995     \else
996       \closein1
997       \begingroup
998         \def\addlanguage#1#2#3#4#5{%
999           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1000             \global\expandafter\let\csname l@#1\endcsname
1001               \csname lang@#1\endcsname
1002           \fi}%
1003         \def\uselanguage#1{%
1004           \input language.def
1005         \endgroup
1006       \fi
1007     \fi
1008     \chardef\l@english\z@
1009 \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.

```

1010 \def\addto#1#2{%
1011   \ifx#1\undefined
1012     \def#1{#2}%
1013   \else
1014     \ifx#1\relax
1015       \def#1{#2}%
1016     \else
1017       {\toks\expandafter{#1#2}%
1018        \xdef#1{\the\toks@}}%
1019   \fi
1020 \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.

```

1021 \bbl@trace{Hooks}
1022 \newcommand\AddBabelHook[3][]{%
1023   \bbl@iifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{%
1024     \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1025     \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1026     \bbl@iifunset{\bbl@ev@#2@#3@#1}%
1027     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1028     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1029     \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1030 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1031 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1032 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1033 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1034   \ifx\UseHook\undefined\else\UseHook{babel/*/#2}\fi
1035   \def\bbl@elth##1{%
1036     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1037     \bbl@cs{ev@#2@#3}%
1038     \ifx\language\undefined\else % Test required for Plain (?)
1039       \ifx\UseHook\undefined\else\UseHook{babel/#1/#2}\fi
1040     \def\bbl@elth##1{%
1041       \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}%
1042       \bbl@cs{ev@#2@#1}%
1043     \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).

```

1044 \def\bbl@evargs{,% <- don't delete this comma
1045   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1046   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1047   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1048   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1049   beforestart=0,language=2,begindocument=1}
1050 \ifx\NewHook\undefined\else % Test for Plain (?)
1051   \def\bbl@tempa#1=#2\@{ \NewHook{babel/#1}}
1052   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@{
1053 \fi

```

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

```

1054 \providecommand\PassOptionsToLocale[2]{%
1055   \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.

```
1056 \bbl@trace{Macros for setting language files up}
1057 \def\bbl@ldfinit{%
1058   \let\bbl@screset\@empty
1059   \let\BabelStrings\bbl@opt@string
1060   \let\BabelOptions\@empty
1061   \let\BabelLanguages\relax
1062   \ifx\originalTeX\@undefined
1063     \let\originalTeX\@empty
1064   \else
1065     \originalTeX
1066   \fi}
1067 \def\LdfInit#1#2{%
1068   \chardef\atcatcode=\catcode`\@
1069   \catcode`\@=11\relax
1070   \chardef\eqcatcode=\catcode`\=
1071   \catcode`\==12\relax
1072   \@ifpackagewith{babel}{ensureinfo=off}}}%
1073   {\ifx\InputIfFileExists\@undefined\else
1074     \bbl@ifunset\bbl@lname@#1}%
1075     {{\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1076       \def\languagename{#1}%
1077       \bbl@id@assign
1078       \bbl@load@info{#1}}}%
1079     {}%
1080   \fi}%
1081   \expandafter\if\expandafter\@backslashchar
1082     \expandafter\@car\string#2\@nil
1083   \ifx#2\@undefined\else
1084     \ldf@quit{#1}%
1085   \fi
1086 \else
1087   \expandafter\ifx\csname#2\endcsname\relax\else
1088     \ldf@quit{#1}%
1089   \fi
1090 \fi
1091 \bbl@ldfinit}
```

\ldf@quit This macro interrupts the processing of a language definition file. Remember \endinput is not executed immediately, but delayed to the end of the current line in the input file.

```
1092 \def\ldf@quit#1{%
1093   \expandafter\main@language\expandafter{#1}%
1094   \catcode`\@=\atcatcode \let\atcatcode\relax
```

```

1095 \catcode`\==\eqcatcode \let\eqcatcode\relax
1096 \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.

```

1097 \def\bbl@afterldf{%
1098 \bbl@afterlang
1099 \let\bbl@afterlang\relax
1100 \let\BabelModifiers\relax
1101 \let\bbl@screset\relax}%
1102 \def\ldf@finish#1{%
1103 \loadlocalcfg{#1}%
1104 \bbl@afterldf
1105 \expandafter\main@language\expandafter{#1}%
1106 \catcode`\@=\atcatcode \let\atcatcode\relax
1107 \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 *L^AT_EX*.

```

1108 \@onlypreamble\LdfInit
1109 \@onlypreamble\ldf@quit
1110 \@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.

```

1111 \def\main@language#1{%
1112 \def\bbl@main@language{#1}%
1113 \let\language\name\bbl@main@language
1114 \let\localename\bbl@main@language
1115 \let\mainlocalename\bbl@main@language
1116 \bbl@id@assign
1117 \bbl@patterns{\language\name}}

```

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

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

```

1118 \def\bbl@beforestart{%
1119 \def\@nolanerr##1{%
1120 \bbl@carg\chardef{l@##1}\z@
1121 \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1122 \bbl@usehooks{beforestart}{}%
1123 \global\let\bbl@beforestart\relax}
1124 \AtBeginDocument{%
1125 {\@nameuse\bbl@beforestart}}% Group!
1126 \if@filesw
1127 \providecommand\babel@aux[2]{}%
1128 \immediate\write\@mainaux{\unexpanded{%
1129 \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1130 \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1131 \fi
1132 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1133 \ifbbl@single % must go after the line above.
1134 \renewcommand\selectlanguage[1]{}%
1135 \renewcommand\foreignlanguage[2]{#2}%
1136 \global\let\babel@aux\@gobbletwo % Also as flag
1137 \fi}

```

```

1138 %
1139 \ifcase\bbl@engine\or
1140   \AtBeginDocument{\pagedir\bodydir}
1141 \fi

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

1142 \def\select@language@x#1{%
1143   \ifcase\bbl@select@type
1144     \bbl@ifsamestring\language#1\{\select@language{#1}}%
1145   \else
1146     \select@language{#1}%
1147 \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.

```

1148 \bbl@trace{Shorhands}
1149 \def\bbl@withactive#1#2{%
1150   \begingroup
1151     \lccode`~=#2\relax
1152     \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.

```

1153 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1154   \bbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1155   \bbl@ifunset{\@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1156   \ifx\nfss@catcodes\undefined\else
1157     \begingroup
1158       \catcode`#1\active
1159       \nfss@catcodes
1160       \ifnum\catcode`#1=\active
1161         \endgroup
1162         \bbl@add\nfss@catcodes{\@makeother#1}%
1163       \else
1164         \endgroup
1165       \fi
1166   \fi}

```

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

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

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

```

1167 \def\bbl@active@def#1#2#3#4{%
1168   \@namedef{#3#1}{%
1169     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1170       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1171     \else
1172       \bbl@afterfi\csname#2@sh@#1\endcsname
1173     \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.

```

1174   \long\@namedef{#3@arg#1}##1{%
1175     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1176       \bbl@afterelse\csname#4#1\endcsname##1%
1177     \else
1178       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1179     \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.

```

1180 \def\initiate@active@char#1{%
1181   \bbl@ifunset{active@char\string#1}%
1182   {\bbl@withactive
1183     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1184   {}}

```

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).

```

1185 \def\@initiate@active@char#1#2#3{%
1186   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1187   \ifx#1\@undefined
1188     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1189   \else
1190     \bbl@csarg\let{oridef@#2}#1%
1191     \bbl@csarg\edef{oridef@#2}{%
1192       \let\noexpand#1%
1193       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1194   \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*").

```

1195   \ifx#1#3\relax
1196     \expandafter\let\csname normal@char#2\endcsname#3%
1197   \else
1198     \bbl@info{Making #2 an active character}%
1199     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1200     \@namedef{normal@char#2}{%
1201       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1202     \else
1203       \@namedef{normal@char#2}{#3}%
1204     \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).

```

1205   \bbl@restoreactive{#2}%
1206   \AtBeginDocument{%

```

```

1207 \catcode`#2\active
1208 \if@filesw
1209 \immediate\write\@mainaux{\catcode`\string#2\active}%
1210 \fi}%
1211 \expandafter\bbbl@add@special\csname#2\endcsname
1212 \catcode`#2\active
1213 \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⟩`).

```

1214 \let\bbbl@tempa\@firstoftwo
1215 \if\string^#2%
1216 \def\bbbl@tempa{\noexpand\textormath}%
1217 \else
1218 \ifx\bbbl@mathnormal\@undefined\else
1219 \let\bbbl@tempa\bbbl@mathnormal
1220 \fi
1221 \fi
1222 \expandafter\edef\csname active@char#2\endcsname{%
1223 \bbbl@tempa
1224 {\noexpand\if@safe@actives
1225 \noexpand\expandafter
1226 \expandafter\noexpand\csname normal@char#2\endcsname
1227 \noexpand\else
1228 \noexpand\expandafter
1229 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1230 \noexpand\fi}%
1231 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1232 \bbbl@csarg\edef{doactive#2}{%
1233 \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

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

(where `\active@char⟨char⟩` is *one* control sequence!).

```

1234 \bbbl@csarg\edef{active@#2}{%
1235 \noexpand\active@prefix\noexpand#1%
1236 \expandafter\noexpand\csname active@char#2\endcsname}%
1237 \bbbl@csarg\edef{normal@#2}{%
1238 \noexpand\active@prefix\noexpand#1%
1239 \expandafter\noexpand\csname normal@char#2\endcsname}%
1240 \bbbl@ncarg\let#1\bbbl@normal@#2}%

```

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

```

1241 \bbbl@active@def#2\user@group{user@active}{language@active}%
1242 \bbbl@active@def#2\language@group{language@active}{system@active}%
1243 \bbbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1244 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1245 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1246 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1247 {\expandafter\noexpand\csname user@active#2\endcsname}%

```


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

```
1248 \if\string'#2%
1249 \let\prim@s\bbl@prim@s
1250 \let\active@math@prime#1%
1251 \fi
1252 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
1253 << *More package options >> ≡
1254 \DeclareOption{math=active}{}
1255 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1256 << /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*.

```
1257 \ifpackagewith{babel}{KeepShorthandsActive}%
1258 {\let\bbl@restoreactive\@gobble}%
1259 {\def\bbl@restoreactive#1{%
1260 \bbl@exp{%
1261 \\\AfterBabelLanguage\\CurrentOption
1262 {\catcode`#1=\the\catcode`#1\relax}%
1263 \\\AtEndOfPackage
1264 {\catcode`#1=\the\catcode`#1\relax}}}%
1265 \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.

```
1266 \def\bbl@sh@select#1#2{%
1267 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1268 \bbl@afterelse\bbl@scndcs
1269 \else
1270 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1271 \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.

```
1272 \begingroup
1273 \bbl@ifunset{ifincsname}
1274 {\gdef\active@prefix#1{%
1275 \ifx\protect\@typeset@protect
1276 \else
1277 \ifx\protect\@unexpandable@protect
1278 \noexpand#1%
1279 \else
1280 \protect#1%
1281 \fi
1282 \expandafter\@gobble
1283 \fi}}
1284 {\gdef\active@prefix#1{%
1285 \ifincsname
```

```

1286     \string#1%
1287     \expandafter\@gobble
1288   \else
1289     \ifx\protect\@typeset@protect
1290     \else
1291       \ifx\protect\@unexpandable@protect
1292         \noexpand#1%
1293       \else
1294         \protect#1%
1295       \fi
1296     \expandafter\expandafter\expandafter\@gobble
1297   \fi
1298 \fi}}
1299 \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 `\if@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`).

```

1300 \newif\if@safe@actives
1301 \@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.

```

1302 \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`.

```

1303 \chardef\bbl@activated\z@
1304 \def\bbl@activate#1{%
1305   \chardef\bbl@activated\@ne
1306   \bbl@withactive{\expandafter\let\expandafter}#1%
1307   \csname bbl@active@string#1\endcsname}
1308 \def\bbl@deactivate#1{%
1309   \chardef\bbl@activated\tw@
1310   \bbl@withactive{\expandafter\let\expandafter}#1%
1311   \csname bbl@normal@string#1\endcsname}

```

\bbl@firstcs

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

```

1312 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1313 \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.

```

1314 \def\babel@texpdf#1#2#3#4{%

```

```

1315 \ifx\texorpdfstring\undefined
1316   \textormath{#1}{#3}%
1317 \else
1318   \texorpdfstring{\textormath{#1}{#3}}{#2}%
1319   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1320 \fi}
1321 %
1322 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1323 \def\@decl@short#1#2#3\@nil#4{%
1324   \def\bbl@tempa{#3}%
1325   \ifx\bbl@tempa\@empty
1326     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1327     \bbl@ifunset{#1@sh@\string#2@}{}%
1328     {\def\bbl@tempa{#4}%
1329       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1330       \else
1331         \bbl@info
1332           {Redefining #1 shorthand \string#2\\%
1333            in language \CurrentOption}%
1334       \fi}%
1335     \@namedef{#1@sh@\string#2@}{#4}%
1336   \else
1337     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1338     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1339     {\def\bbl@tempa{#4}%
1340       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1341       \else
1342         \bbl@info
1343           {Redefining #1 shorthand \string#2\string#3\\%
1344            in language \CurrentOption}%
1345       \fi}%
1346     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1347   \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.

```

1348 \def\textormath{%
1349   \ifmmode
1350     \expandafter\@secondoftwo
1351   \else
1352     \expandafter\@firstoftwo
1353   \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’.

```

1354 \def\user@group{user}
1355 \def\language@group{english}
1356 \def\system@group{system}

```

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

```

1357 \def\useshorthands{%
1358   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1359 \def\bbl@usesh@s#1{%
1360   \bbl@usesh@x
1361   {\AddBabelHook{babel-sh-\string#1}{afterextras}}{\bbl@activate{#1}}}%
1362   {#1}}

```

```

1363 \def\bbl@usesh@x#1#2{%
1364   \bbl@ifshorthand{#2}%
1365   {\def\user@group{user}%
1366    \initiate@active@char{#2}%
1367    #1%
1368    \bbl@activate{#2}}%
1369   {\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.

```

1370 \def\user@language@group{user@\language@group}
1371 \def\bbl@set@user@generic#1#2{%
1372   \bbl@ifunset{user@generic@active#1}%
1373   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1374    \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1375    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1376     \expandafter\noexpand\csname normal@char#1\endcsname}%
1377    \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1378     \expandafter\noexpand\csname user@active#1\endcsname}%
1379    \@empty}
1380 \newcommand\defineshorthand[3][user]{%
1381   \edef\bbl@tempa{\zap@space#1 \@empty}%
1382   \bbl@for\bbl@tempb\bbl@tempa{%
1383     \if*\expandafter\@car\bbl@tempb\@nil
1384     \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1385     \@expandtwoargs
1386     \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1387   \fi
1388   \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.

```

1389 \def\languageshorthands#1{%
1390   \bbl@ifsamestring{none}{#1}{}%
1391   \bbl@once{short-\localename-#1}{%
1392     \bbl@info{'\localename' activates '#1' shorthands.\Reported}}}%
1393   \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`.

```

1394 \def\aliasshorthand#1#2{%
1395   \bbl@ifshorthand{#2}%
1396   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1397     \ifx\document@notprerr
1398       \notshorthand{#2}%
1399     \else
1400       \initiate@active@char{#2}%
1401       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1402       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1403       \bbl@activate{#2}%
1404     \fi
1405   \fi}%
1406   {\bbl@error{shorthand-is-off}{#2}{}}}

```

\@notshorthand

```

1407 \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.

```
1408 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1409 \DeclareRobustCommand*\shorthandoff{%
1410   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1411 \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.

```
1412 \def\bbl@switch@sh#1#2{%
1413   \ifx#2\@nnil\else
1414     \bbl@ifunset{bbl@active@\string#2}%
1415     {\bbl@error{not-a-shorthand-b}{\string#2}}}%
1416     {\ifcase#1%   off, on, off*
1417       \catcode`#2\relax
1418       \or
1419       \catcode`#2\active
1420       \bbl@ifunset{bbl@shdef@\string#2}%
1421       {}%
1422       {\bbl@withactive{\expandafter\let\expandafter}#2%
1423         \csname bbl@shdef@\string#2\endcsname
1424         \bbl@csarg\let{shdef@\string#2}\relax}%
1425       \ifcase\bbl@activated\or
1426       \bbl@activate{#2}%
1427       \else
1428       \bbl@deactivate{#2}%
1429       \fi
1430       \or
1431       \bbl@ifunset{bbl@shdef@\string#2}%
1432       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1433       {}%
1434       \csname bbl@oricat@\string#2\endcsname
1435       \csname bbl@oridef@\string#2\endcsname
1436       \fi}%
1437   \bbl@afterfi\bbl@switch@sh#1%
1438   \fi}
```

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

```
1439 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1440 \def\bbl@putsh#1{%
1441   \bbl@ifunset{bbl@active@\string#1}%
1442   {\bbl@putsh@i#1\@empty\@nnil}%
1443   {\csname bbl@active@\string#1\endcsname}}
1444 \def\bbl@putsh@i#1#2\@nnil{%
1445   \csname\language@group @sh@\string#1@%
1446     \ifx\@empty#2\else\string#2@\fi\endcsname}
1447 %
1448 \ifx\bbl@opt@shorthands\@nnil\else
1449   \let\bbl@s@initiate@active@char\initiate@active@char
1450   \def\initiate@active@char#1{%
1451     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1452   \let\bbl@s@switch@sh\bbl@switch@sh
1453   \def\bbl@switch@sh#1#2{%
1454     \ifx#2\@nnil\else
```

```

1455 \bbl@afterfi
1456 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1457 \fi}
1458 \let\bbl@s@activate\bbl@activate
1459 \def\bbl@activate#1{%
1460 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1461 \let\bbl@s@deactivate\bbl@deactivate
1462 \def\bbl@deactivate#1{%
1463 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1464 \fi

```

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

```

1465 \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.

```

1466 \def\bbl@prim@s{%
1467 \prime\futurelet\@let@token\bbl@pr@m@s}
1468 \def\bbl@if@primes#1#2{%
1469 \ifx#1\@let@token
1470 \expandafter\@firstoftwo
1471 \else\ifx#2\@let@token
1472 \bbl@afterelse\expandafter\@firstoftwo
1473 \else
1474 \bbl@afterfi\expandafter\@secondoftwo
1475 \fi\fi}
1476 \begingroup
1477 \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1478 \catcode`\'=12 \catcode`\"=\active \lccode`\"='\ '
1479 \lowercase{%
1480 \gdef\bbl@pr@m@s{%
1481 \bbl@if@primes" '%
1482 \pr@@@s
1483 {\bbl@if@primes*\^pr@@@t\egroup}}}
1484 \endgroup

```

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

```

1485 \initiate@active@char{~}
1486 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1487 \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.

```

1488 \expandafter\def\csname OT1dqpos\endcsname{127}
1489 \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

```

1490 \ifx\f@encoding\undefined
1491 \def\f@encoding{OT1}
1492 \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.

```
1493 \bbl@trace{Language attributes}
1494 \newcommand\languageattribute[2]{%
1495   \def\bbl@tempc{#1}%
1496   \bbl@fixname\bbl@tempc
1497   \bbl@iflanguage\bbl@tempc{%
1498     \bbl@vforeach{#2}{%
```

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

```
1499     \ifx\bbl@known@attrs\undefined
1500       \in@false
1501     \else
1502       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1503     \fi
1504     \ifin@
1505       \bbl@warning{%
1506         You have more than once selected the attribute '##1'\%
1507         for language #1. Reported}%
1508     \else
```

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

```
1509       \bbl@info{Activated '##1' attribute for\%
1510         '\bbl@tempc'. Reported}%
1511       \bbl@exp{%
1512         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1513       \edef\bbl@tempa{\bbl@tempc-##1}%
1514       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1515       {\csname\bbl@tempc @attr##1\endcsname}%
1516       {\@attrerr{\bbl@tempc}{##1}}%
1517     \fi}}
1518 \@onlypreamble\languageattribute
```

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

```
1519 \newcommand*\@attrerr[2]{%
1520   \bbl@error{unknown-attribute}{#1}{#2}{}}
```

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

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

```
1521 \def\bbl@declare@ttribute#1#2#3{%
1522   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1523   \ifin@
1524     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1525   \fi
1526   \bbl@add@list\bbl@attributes{#1-#2}%
1527   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret \TeX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```
1528 \def\bbl@ifattributeset#1#2#3#4{%
1529   \ifx\bbl@known@attrs\@undefined
1530     \in@false
1531   \else
1532     \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1533   \fi
1534   \ifin@
1535     \bbl@afterelse#3%
1536   \else
1537     \bbl@afterfi#4%
1538   \fi}
```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise.

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

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

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

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

4.10. Support for saving and redefining macros

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`). Note undefined macros are not undefined any more when saved – they are *\relax*'ed.

\babel@savecnt

\babel@beginsave The initialization of a new save cycle: reset the counter to zero.

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

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

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


\babel@save

\babel@savevariable The macro `\babel@save<csname>` saves the current meaning of the control sequence `<csname>` to `\originalTeX` (which has to be expandable, i.e., you shouldn't let it to `\relax`). To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented. The macro `\babel@savevariable<variable>` saves the value of the variable. `<variable>` can be anything allowed after the `\the` primitive. To avoid messing saved definitions up, they are saved only the very first time.

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

\bbl@redefine To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don't want to redefine the \TeX macros completely in case their definitions change (they have changed in the past). A macro named `\macro` will be saved new control sequences named `\org@macro`.

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

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

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

\bbl@redefineroobust For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo_`. So it is necessary to check whether `\foo_` exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define `\foo_`.

```
1587 \def\bbl@redefineroobust#1{%
1588   \edef\bbl@tempa{\bbl@stripslash#1}%
1589   \bbl@ifunset{\bbl@tempa\space}%
1590   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1591     \bbl@exp{\def\\#1{\protect<\bbl@tempa\space>}}}%
1592   {\bbl@exp{\let<org@\bbl@tempa><\bbl@tempa\space>}}}%
1593   \@namedef{\bbl@tempa\space}}
1594 \@onlypreamble\bbl@redefineroobust
```

4.11. French spacing

\bbl@frenchspacing

\bbl@nonfrenchspacing Some languages need to have \frenchspacing in effect. Others don't want that. The command \bbl@frenchspacing switches it on when it isn't already in effect and \bbl@nonfrenchspacing switches it off if necessary.

```

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

```

A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in \babelprovide. This new method should be ideally the default one.

```

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

```

4.12. Hyphens

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

```

1631 \bbl@trace{Hyphens}
1632 \@onlypreamble\babelhyphenation
1633 \AtEndOfPackage{%
1634   \newcommand\babelhyphenation[2][\@empty]{%
1635     \ifx\bbl@hyphenation@\relax
1636       \let\bbl@hyphenation@\@empty
1637     \fi
1638     \ifx\bbl@hyphlist\@empty\else
1639       \bbl@warning{%
1640         You must not intermingle \string\selectlanguage\space and\\%
1641         \string\babelhyphenation\space or some exceptions will not\\%
1642         be taken into account. Reported}%
1643       \fi

```

```

1644 \ifx\@empty#1%
1645 \protected@edef\bb@hyphenation@\bb@hyphenation\space#2}%
1646 \else
1647 \bb@vforeach{#1}{%
1648 \def\bb@tempa{##1}%
1649 \bb@fixname\bb@tempa
1650 \bb@iflanguage\bb@tempa{%
1651 \bb@csarg\protected@edef\hyphenation@\bb@tempa}{%
1652 \bb@ifunset\bb@hyphenation@\bb@tempa}%
1653 }%
1654 {\csname \bb@hyphenation@\bb@tempa\endcsname\space}%
1655 #2}}}%
1656 \fi}}

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

1685 \def\bb@usehyphen#1{%
1686 \leavevmode

```

```

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

```

The following macro inserts the hyphen char.

```

1691 \def\bbl@hyphenchar{%
1692 \ifnum\hyphenchar\font=\m@ne
1693 \babelnullhyphen
1694 \else
1695 \char\hyphenchar\font
1696 \fi}

```

Finally, we define the hyphen “types”. Their names will not change, so you may use them in `ldf`’s. After a space, the `\mbox` in `\bbl@hy@nobreak` is redundant.

```

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

```

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

```

1711 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}

```

4.13. Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by `luatex` and `xetex`. The code is organized here with pseudo-guards, so we start with the basic commands.

Tools But first, a tool. It makes global a local variable. This is not the best solution, but it works.

```

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

```

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

```

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

```

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

```

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

```

Main command This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

```

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

```

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

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

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

Now we define commands to be used inside \StartBabelCommands.

Strings The following macro is the actual definition of \SetString when it is “active”

First save the “switcher”. Create it if undefined. Strings are defined only if undefined (i.e., like \providescommand). With the event stringprocess you can preprocess the string by manipulating the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in the same order as defined. Finally, the string is set.

```

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

```

A little auxiliary command sets the string. Formerly used with casing. Very likely no longer necessary, although it’s used in \setlocalecaption.

```

1857 \def\bbl@scset#1#2{\def#1{#2}}

```

Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current definition is somewhat complicated because we need a count, but \count@ is not under our control (remember \SetString may call hooks). Instead of defining a dedicated count, we just “pre-expand” its value.

```

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

```

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

```

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

```

Case mapping The command `\SetCase` is deprecated. Currently it consists in a definition with a hack just for backward compatibility in the macro mapping.

```

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

```

Macros to deal with case mapping for hyphenation. To decide if the document is monolingual or multilingual, we make a rough guess – just see if there is a comma in the languages list, built in the first pass of the package options.

```

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

```

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

```

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

```

The following package options control the behavior of hyphenation mapping.

```

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

```


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

```

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

```

4.14. Tailor captions

A general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```

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

```

```

1980      \bbl@tglobal\bbl@captionslist
1981      \fi
1982      \fi}

```

4.15. Making glyphs available

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

\set@low@box The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```

1983 \bbl@trace{Macros related to glyphs}
1984 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1985      \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
1986      \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.

```

1987 \def\save@sf@q#1{\leavevmode
1988      \begingroup
1989      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1990      \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.

```

1991 \ProvideTextCommand{\quotedblbase}{OT1}{%
1992      \save@sf@q{\set@low@box{\textquotedblright/}}%
1993      \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

1994 \ProvideTextCommandDefault{\quotedblbase}{%
1995      \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1996 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1997      \save@sf@q{\set@low@box{\textquoteright/}}%
1998      \box\z@\kern-.04em\bbl@allowhyphens}}

```

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

```

1999 \ProvideTextCommandDefault{\quotesinglbase}{%
2000      \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

2001 \ProvideTextCommand{\guillemetleft}{OT1}{%
2002      \ifmmode
2003          \ll
2004      \else
2005          \save@sf@q{\nobreak
2006              \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2007          \fi}
2008 \ProvideTextCommand{\guillemetright}{OT1}{%
2009      \ifmmode
2010          \gg
2011      \else
2012          \save@sf@q{\nobreak
2013              \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%

```

```

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

```

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

```

2029 \ProvideTextCommandDefault{\guillemetleft}{%
2030 \UseTextSymbol{OT1}{\guillemetleft}}
2031 \ProvideTextCommandDefault{\guillemetright}{%
2032 \UseTextSymbol{OT1}{\guillemetright}}
2033 \ProvideTextCommandDefault{\guillemotleft}{%
2034 \UseTextSymbol{OT1}{\guillemotleft}}
2035 \ProvideTextCommandDefault{\guillemotright}{%
2036 \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2037 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2038 \ifmmode
2039 <%
2040 \else
2041 \save@sf@q{\nobreak
2042 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2043 \fi}
2044 \ProvideTextCommand{\guilsinglright}{OT1}{%
2045 \ifmmode
2046 >%
2047 \else
2048 \save@sf@q{\nobreak
2049 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2050 \fi}

```

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

```

2051 \ProvideTextCommandDefault{\guilsinglleft}{%
2052 \UseTextSymbol{OT1}{\guilsinglleft}}
2053 \ProvideTextCommandDefault{\guilsinglright}{%
2054 \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.

```

2055 \DeclareTextCommand{\ij}{OT1}{%
2056 i\kern-0.02em\bbl@allowhyphens j}
2057 \DeclareTextCommand{\IJ}{OT1}{%
2058 I\kern-0.02em\bbl@allowhyphens J}
2059 \DeclareTextCommand{\ij}{T1}{\char188}
2060 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2061 \ProvideTextCommandDefault{\ij}{%
2062   \UseTextSymbol{OT1}{\ij}}
2063 \ProvideTextCommandDefault{\IJ}{%
2064   \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).

```
2065 \def\crrtic@{\hrule height0.1ex width0.3em}
2066 \def\crttic@{\hrule height0.1ex width0.33em}
2067 \def\ddj@{%
2068   \setbox0\hbox{d}\dimen@=\ht0
2069   \advance\dimen@lex
2070   \dimen@.45\dimen@
2071   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2072   \advance\dimen@ii.5ex
2073   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2074 \def\DDJ@{%
2075   \setbox0\hbox{D}\dimen@=.55\ht0
2076   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2077   \advance\dimen@ii.15ex % correction for the dash position
2078   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2079   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2080   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2081 %
2082 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2083 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2084 \ProvideTextCommandDefault{\dj}{%
2085   \UseTextSymbol{OT1}{\dj}}
2086 \ProvideTextCommandDefault{\DJ}{%
2087   \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.

```
2088 \DeclareTextCommand{\SS}{OT1}{SS}
2089 \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.

```
2090 \ProvideTextCommandDefault{\glq}{%
2091   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2092 \ProvideTextCommand{\grq}{T1}{%
2093   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2094 \ProvideTextCommand{\grq}{TU}{%
2095   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2096 \ProvideTextCommand{\grq}{OT1}{%
2097   \save@sf@q{\kern-.0125em
2098     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2099   }
```

```

2099 \kern.07em\relax}}
2100 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2101 \ProvideTextCommandDefault{\glqq}{%
2102 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2103 \ProvideTextCommand{\grqq}{T1}{%
2104 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2105 \ProvideTextCommand{\grqq}{TU}{%
2106 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2107 \ProvideTextCommand{\grqq}{0T1}{%
2108 \save@sf@q{\kern-.07em
2109 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2110 \kern.07em\relax}}
2111 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2112 \ProvideTextCommandDefault{\flq}{%
2113 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2114 \ProvideTextCommandDefault{\frq}{%
2115 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```

2116 \ProvideTextCommandDefault{\flqq}{%
2117 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2118 \ProvideTextCommandDefault{\frqq}{%
2119 \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).

```

2120 \def\umlauthigh{%
2121 \def\bbl@umlauta##1{\leavevmode\bgroup%
2122 \accent\csname\f@encoding dqpos\endcsname
2123 ##1\bbl@allowhyphens\egroup}%
2124 \let\bbl@umlaute\bbl@umlauta}
2125 \def\umlautlow{%
2126 \def\bbl@umlauta{\protect\lower@umlaut}}
2127 \def\umlautelow{%
2128 \def\bbl@umlaute{\protect\lower@umlaut}}
2129 \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.

```
2130 \expandafter\ifx\csname U@D\endcsname\relax
2131 \csname newdimen\endcsname U@D
2132 \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.

```
2133 \def\lower@umlaut#1{%
2134 \leavevmode\bgroup
2135 \U@D lex%
2136 {\setbox\z@\hbox{%
2137 \char\csname f@encoding dqpos\endcsname}%
2138 \dimen@ -.45ex\advance\dimen@ \ht\z@
2139 \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2140 \accent\csname f@encoding dqpos\endcsname
2141 \fontdimen5\font\U@D #1%
2142 \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).

```
2143 \AtBeginDocument{%
2144 \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2145 \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2146 \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2147 \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2148 \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2149 \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2150 \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2151 \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2152 \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2153 \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2154 \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}}
```

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

```
2155 \ifx\l@english\@undefined
2156 \chardef\l@english\z@
2157 \fi
2158 % The following is used to cancel rules in ini files (see Amharic).
2159 \ifx\l@unhyphenated\@undefined
2160 \newlanguage\l@unhyphenated
2161 \fi
```

4.16. Layout

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

```
2162 \bbl@trace{Bidi layout}
2163 \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.

```
2164 \bbl@trace{Input engine specific macros}
2165 \ifcase\bbl@engine
2166   \input txtbabel.def
2167 \or
2168   \input luababel.def
2169 \or
2170   \input xebabel.def
2171 \fi
2172 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2173 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2174 \ifx\babelposthyphenation\undefined
2175   \let\babelposthyphenation\babelprehyphenation
2176   \let\babelpatterns\babelprehyphenation
2177   \let\babelcharproperty\babelprehyphenation
2178 \fi
2179 \end{package} | core
```

4.18. Creating and modifying languages

Continue with \LaTeX only.

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

```
2180 \begin{package}
2181 \bbl@trace{Creating languages and reading ini files}
2182 \let\bbl@extend@ini@gobble
2183 \newcommand\babelprovide[2][]{%
2184   \let\bbl@savelangname\languagename
2185   \edef\bbl@savelocaleid{\the\localeid}%
2186   % Set name and locale id
2187   \edef\languagename{#2}%
2188   \bbl@id@assign
2189   % Initialize keys
2190   \bbl@vforeach{captions,date,import,main,script,language,%
2191     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2192     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2193     Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2194     {\bbl@csarg\let{KVP@##1}\@nnil}%
2195   \global\let\bbl@release@transforms\@empty
2196   \global\let\bbl@release@casing\@empty
2197   \let\bbl@calendars\@empty
2198   \global\let\bbl@inidata\@empty
2199   \global\let\bbl@extend@ini@gobble
2200   \global\let\bbl@included@inis\@empty
2201   \gdef\bbl@key@list{;}%
2202   \bbl@ifunset{bbl@passto@#2}%
2203     {\def\bbl@tempa{#1}}%
2204     {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}}%
2205   \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2206     \in@{/}{##1}% With /, (re)sets a value in the ini
2207     \ifin@
2208       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2209       \bbl@renewinikey##1\@{##2}%
2210     \else
2211       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2212         \bbl@error{unknown-provide-key}{##1}{}{}%
2213       \fi
2214       \bbl@csarg\def{KVP@##1}{##2}%
2215     \fi}%
\end{package}
```

```

2216 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2217 \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw}%
2218 % == init ==
2219 \ifx\bbl@screset\@undefined
2220 \bbl@ldfinit
2221 \fi
2222 % ==
2223 % If there is no import (last wins), use @import (internal, there
2224 % must be just one). To consider any order (because
2225 % \PassOptionsToLocale).
2226 \ifx\bbl@KVP@import\@nnil
2227 \let\bbl@KVP@import\bbl@KVP@@import
2228 \fi
2229 % == date (as option) ==
2230 % \ifx\bbl@KVP@date\@nnil\else
2231 % \fi
2232 % ==
2233 \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2234 \ifcase\bbl@howloaded
2235 \let\bbl@lbkflag\@empty % new
2236 \else
2237 \ifx\bbl@KVP@hyphenrules\@nnil\else
2238 \let\bbl@lbkflag\@empty
2239 \fi
2240 \ifx\bbl@KVP@import\@nnil\else
2241 \let\bbl@lbkflag\@empty
2242 \fi
2243 \fi
2244 % == import, captions ==
2245 \ifx\bbl@KVP@import\@nnil\else
2246 \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
2247 {\ifx\bbl@initoload\relax
2248 \begingroup
2249 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2250 \bbl@input@texini{#2}%
2251 \endgroup
2252 \else
2253 \xdef\bbl@KVP@import{\bbl@initoload}%
2254 \fi}%
2255 {}%
2256 \let\bbl@KVP@date\@empty
2257 \fi
2258 \let\bbl@KVP@captions@@\bbl@KVP@captions
2259 \ifx\bbl@KVP@captions\@nnil
2260 \let\bbl@KVP@captions\bbl@KVP@import
2261 \fi
2262 % ==
2263 \ifx\bbl@KVP@transforms\@nnil\else
2264 \bbl@replace\bbl@KVP@transforms{ }{,}%
2265 \fi
2266 % == Load ini ==
2267 \ifcase\bbl@howloaded
2268 \bbl@provide@new{#2}%
2269 \else
2270 \bbl@ifblank{#1}%
2271 {}% With \bbl@load@basic below
2272 {\bbl@provide@renew{#2}}%
2273 \fi
2274 % Post tasks
2275 % -----
2276 % == subsequent calls after the first provide for a locale ==
2277 \ifx\bbl@inidata\@empty\else
2278 \bbl@extend@ini{#2}%

```



```

2279 \fi
2280 % == ensure captions ==
2281 \ifx\bbbl@KVP@captions\@nnil\else
2282   \bbbl@ifunset{bbbl@extracaps@#2}%
2283     {\bbbl@exp{\bbabelensure[exclude=\\today]{#2}}}%
2284     {\bbbl@exp{\bbabelensure[exclude=\\today,
2285       include=\bbbl@extracaps@#2]}{#2}}%
2286   \bbbl@ifunset{bbbl@ensure@\language\name}%
2287     {\bbbl@exp{%
2288       \\DeclareRobustCommand\<bbbl@ensure@\language\name>[1]{%
2289         \\foreignlanguage{\language\name}%
2290         {###1}}}%
2291     }%
2292   \bbbl@exp{%
2293     \\bbbl@tglobal\<bbbl@ensure@\language\name>%
2294     \\bbbl@tglobal\<bbbl@ensure@\language\name\space>%
2295 \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.

```

2296 \bbbl@load@basic{#2}%
2297 % == script, language ==
2298 % Override the values from ini or defines them
2299 \ifx\bbbl@KVP@script\@nnil\else
2300   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2301 \fi
2302 \ifx\bbbl@KVP@language\@nnil\else
2303   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2304 \fi
2305 \ifcase\bbbl@engine\or
2306   \bbbl@ifunset{bbbl@chrng@\language\name}{%
2307     {\directlua{
2308       Babel.set_chrngs_b('\bbbl@cl{sbc}', '\bbbl@cl{chrng}') }}%
2309 \fi
2310 % == Line breaking: intraspace, intrapenalty ==
2311 % For CJK, East Asian, Southeast Asian, if interspace in ini
2312 \ifx\bbbl@KVP@intraspace\@nnil\else % We can override the ini or set
2313   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2314 \fi
2315 \bbbl@provide@intraspace
2316 % == Line breaking: justification ==
2317 \ifx\bbbl@KVP@justification\@nnil\else
2318   \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2319 \fi
2320 \ifx\bbbl@KVP@linebreaking\@nnil\else
2321   \bbbl@xin@{\bbbl@KVP@linebreaking,%
2322     {,elongated,kashida,cjk,padding,unhyphenated},%
2323   \ifin@
2324     \bbbl@csarg\xdef
2325       {lnbrk@\language\name}{\expandafter\@car\bbbl@KVP@linebreaking\@nil}%
2326 \fi
2327 \fi
2328 \bbbl@xin@{/e}{\bbbl@cl{lnbrk}}%
2329 \ifin@\else\bbbl@xin@{/k}{\bbbl@cl{lnbrk}}\fi
2330 \ifin@\bbbl@arabicjust\fi
2331 \bbbl@xin@{/p}{\bbbl@cl{lnbrk}}%
2332 \ifin@\AtBeginDocument{\@nameuse{bbbl@tibetanjust}}\fi
2333 % == Line breaking: hyphenate.other.(locale|script) ==
2334 \ifx\bbbl@lbfkflag\@empty
2335   \bbbl@ifunset{bbbl@hyotl@\language\name}{%
2336     {\bbbl@csarg\bbbl@replace{hyotl@\language\name}{ }{,}%
2337       \bbbl@startcommands*\language\name}%

```

```

2338     \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2339     \ifcase\bbl@engine
2340     \ifnum##1<257
2341     \SetHyphenMap{\BabelLower{##1}{##1}}%
2342     \fi
2343     \else
2344     \SetHyphenMap{\BabelLower{##1}{##1}}%
2345     \fi}%
2346     \bbl@endcommands}%
2347 \bbl@ifunset{bbl@hyots@\languagename}{}%
2348 {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2349 \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2350 \ifcase\bbl@engine
2351 \ifnum##1<257
2352 \global\lccode##1=##1\relax
2353 \fi
2354 \else
2355 \global\lccode##1=##1\relax
2356 \fi}}%
2357 \fi
2358 % == Counters: maparabic ==
2359 % Native digits, if provided in ini (TeX level, xe and lua)
2360 \ifcase\bbl@engine\else
2361 \bbl@ifunset{bbl@dgnat@\languagename}{}%
2362 {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2363 \expandafter\expandafter\expandafter
2364 \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2365 \ifx\bbl@KVP@maparabic\@nnil\else
2366 \ifx\bbl@latinarabic\@undefined
2367 \expandafter\let\expandafter\@arabic
2368 \csname bbl@counter@\languagename\endcsname
2369 \else % i.e., if layout=counters, which redefines \@arabic
2370 \expandafter\let\expandafter\bbl@latinarabic
2371 \csname bbl@counter@\languagename\endcsname
2372 \fi
2373 \fi
2374 \fi}%
2375 \fi
2376 % == Counters: mapdigits ==
2377 % > luababel.def
2378 % == Counters: alph, Alph ==
2379 \ifx\bbl@KVP@alph\@nnil\else
2380 \bbl@exp{%
2381   \\\bbl@add\<bbl@preextras@\languagename>{%
2382     \\\babel@save\\\@alph
2383     \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2384 \fi
2385 \ifx\bbl@KVP@Alph\@nnil\else
2386 \bbl@exp{%
2387   \\\bbl@add\<bbl@preextras@\languagename>{%
2388     \\\babel@save\\\@Alph
2389     \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2390 \fi
2391 % == Casing ==
2392 \bbl@release@casing
2393 \ifx\bbl@KVP@casing\@nnil\else
2394 \bbl@csarg\xdef{casing@\languagename}%
2395 {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2396 \fi
2397 % == Calendars ==
2398 \ifx\bbl@KVP@calendar\@nnil
2399 \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2400 \fi

```

```

2401 \def\bbl@tempe##1 ##2\@{% Get first calendar
2402   \def\bbl@tempa{##1}%
2403   \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\\@}%
2404 \def\bbl@tempe##1.##2.##3\@{%
2405   \def\bbl@tempc{##1}%
2406   \def\bbl@tempb{##2}%
2407 \expandafter\bbl@tempe\bbl@tempa..\@@
2408 \bbl@csarg\edef{calpr@\language\name}{%
2409   \ifx\bbl@tempc\empty\else
2410     calendar=\bbl@tempc
2411   \fi
2412   \ifx\bbl@tempb\empty\else
2413     ,variant=\bbl@tempb
2414   \fi}%
2415 % == engine specific extensions ==
2416 % Defined in XXXbabel.def
2417 \bbl@provide@extra{#2}%
2418 % == require.babel in ini ==
2419 % To load or reload the babel-*.tex, if require.babel in ini
2420 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2421   \bbl@ifunset{\bbl@rqtex@\language\name}{}%
2422     {\expandafter\ifx\csname\bbl@rqtex@\language\name\endcsname\empty\else
2423       \let\BabelBeforeIni@gobbletwo
2424       \chardef\atcatcode=\catcode\@
2425       \catcode\@=11\relax
2426       \def\CurrentOption{#2}%
2427       \bbl@input@texini{\bbl@cs{rqtex@\language\name}}%
2428       \catcode\@=\atcatcode
2429       \let\atcatcode\relax
2430       \global\bbl@csarg\let{rqtex@\language\name}\relax
2431     \fi}%
2432 \bbl@foreach\bbl@calendars{%
2433   \bbl@ifunset{\bbl@ca##1}{%
2434     \chardef\atcatcode=\catcode\@
2435     \catcode\@=11\relax
2436     \InputIfFileExists{babel-ca-##1.tex}{}}}%
2437     \catcode\@=\atcatcode
2438     \let\atcatcode\relax}%
2439   {}}%
2440 \fi
2441 % == frenchspacing ==
2442 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2443 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2444 \ifin@
2445   \bbl@extras@wrap{\bbl@pre@fs}%
2446   {\bbl@pre@fs}%
2447   {\bbl@post@fs}%
2448 \fi
2449 % == transforms ==
2450 % > luababel.def
2451 \def\CurrentOption{#2}%
2452 \@nameuse{\bbl@icsave@#2}%
2453 % == main ==
2454 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2455   \let\language\bbl@savelangname
2456   \chardef\localeid\bbl@savelocaleid\relax
2457 \fi
2458 % == hyphenrules (apply if current) ==
2459 \ifx\bbl@KVP@hyphenrules\@nnil\else
2460   \ifnum\bbl@savelocaleid=\localeid
2461     \language\@nameuse{l@\language\name}%
2462   \fi
2463 \fi}

```

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

```

2464 \def\bbl@provide@new#1{%
2465   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2466   \@namedef{extras#1}{}%
2467   \@namedef{noextras#1}{}%
2468   \bbl@startcommands*{#1}{captions}%
2469   \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2470     \def\bbl@tempb##1{% elt for \bbl@captionslist
2471       \ifx##1\@nnil\else
2472         \bbl@exp{%
2473           \\SetString\\##1{%
2474             \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2475           \expandafter\bbl@tempb
2476         \fi}%
2477     \expandafter\bbl@tempb\bbl@captionslist\@nnil
2478   \else
2479     \ifx\bbl@initoload\relax
2480       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2481     \else
2482       \bbl@read@ini{\bbl@initoload}2% % Same
2483     \fi
2484   \fi
2485   \StartBabelCommands*{#1}{date}%
2486   \ifx\bbl@KVP@date\@nnil
2487     \bbl@exp{%
2488       \\SetString\\today{\\bbl@nocaption{today}{#1today}}}%
2489   \else
2490     \bbl@savetoday
2491     \bbl@savedate
2492   \fi
2493   \bbl@endcommands
2494   \bbl@load@basic{#1}%
2495   % == hyphenmins == (only if new)
2496   \bbl@exp{%
2497     \gdef\<#1hyphenmins>{%
2498       {\bbl@ifunset{\bbl@lfthm#1}{2}{\bbl@cs{lfthm#1}}}%
2499       {\bbl@ifunset{\bbl@rgthm#1}{3}{\bbl@cs{rgthm#1}}}}}%
2500   % == hyphenrules (also in renew) ==
2501   \bbl@provide@hyphens{#1}%
2502   \ifx\bbl@KVP@main\@nnil\else
2503     \expandafter\main@language\expandafter{#1}%
2504   \fi}
2505 %
2506 \def\bbl@provide@renew#1{%
2507   \ifx\bbl@KVP@captions\@nnil\else
2508     \StartBabelCommands*{#1}{captions}%
2509     \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2510     \EndBabelCommands
2511   \fi
2512   \ifx\bbl@KVP@date\@nnil\else
2513     \StartBabelCommands*{#1}{date}%
2514     \bbl@savetoday
2515     \bbl@savedate
2516     \EndBabelCommands
2517   \fi
2518   % == hyphenrules (also in new) ==
2519   \ifx\bbl@lbkflag\@empty
2520     \bbl@provide@hyphens{#1}%
2521   \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.

```

2522 \def\bbl@load@basic#1{%
2523   \ifcase\bbl@howloaded\or\or
2524     \ifcase\csname bbl@llevel@\languagename\endcsname
2525       \bbl@csarg\let{lname@\languagename}\relax
2526     \fi
2527   \fi
2528   \bbl@ifunset\bbl@lname@#1{%
2529     {\def\BabelBeforeIni##1##2{%
2530       \begingroup
2531         \let\bbl@inicalptions@aux\@gobbletwo
2532         \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{%
2533           \bbl@read@ini{##1}l%
2534           \ifx\bbl@initoload\relax\endinput\fi
2535         \endgroup}%
2536       \begingroup      % boxed, to avoid extra spaces:
2537       \ifx\bbl@initoload\relax
2538         \bbl@input@texini{##1}%
2539       \else
2540         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2541       \fi
2542     \endgroup}%
2543   {}}

```

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).

```

2544 \def\bbl@load@info#1{%
2545   \def\BabelBeforeIni##1##2{%
2546     \begingroup
2547       \bbl@read@ini{##1}0%
2548       \endinput      % babel- .tex may contain only preamble's
2549     \endgroup}%      boxed, to avoid extra spaces:
2550   {\bbl@input@texini{##1}}

```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```

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

```

```

2576         {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2577         }%                               if hyphenrules found:
2578         {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2579     \fi
2580 \fi
2581 \bbl@ifunset{l@#1}%
2582     {\ifnum\@tempcnta=\m@ne
2583       \bbl@carg\adddialect{l@#1}\language
2584     \else
2585       \bbl@carg\adddialect{l@#1}\@tempcnta
2586     \fi}%
2587     {\ifnum\@tempcnta=\m@ne\else
2588       \global\bbl@carg\chardef{l@#1}\@tempcnta
2589     \fi}}

```

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

```

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

```

The following macros read and store ini files (but don't process them). For each line, there are 3 possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbl@read@ini.

```

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

```

4.19. Main loop in 'provide'

Now, the 'main loop', \bbl@read@ini, which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps:

first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, ‘export’ some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 (without import) or 2 (which import). The value –1 is used with \DocumentMetadata.

\bbl@loop@ini is the reader, line by line (1: stream), and calls \bbl@iniline to save the key/value pairs. If \bbl@inistore finds the @include directive, the input stream is switched temporarily and \bbl@read@subini is called.

When the language is being set based on the document metadata (#2 in \bbl@read@ini is –1), there is an interlude to get the name, after the data have been collected, and before it's processed.

```

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

```

```

2680 \def\bbl@tempa##1 ##2\@{##1}% Get first name
2681 \def\bbl@elt##1##2##3{%
2682 \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2683 {\edef\language\language{\bbl@tempa##3 \@@}%
2684 \bbl@id@assign
2685 \def\bbl@elt####1####2####3{}}%
2686 {}}%
2687 \bbl@inidata
2688 \fi
2689 \bbl@csarg\xdef{\l@ini\language}{#1}%
2690 \bbl@read@ini@aux
2691 % == 'Export' data ==
2692 \bbl@ini@exports{#2}%
2693 \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2694 \global\let\bbl@inidata\@empty
2695 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language}}%
2696 \bbl@tglobal\bbl@ini@loaded
2697 \fi
2698 \closein\bbl@readstream}
2699 \def\bbl@read@ini@aux{%
2700 \let\bbl@savestrings\@empty
2701 \let\bbl@savetoday\@empty
2702 \let\bbl@savestate\@empty
2703 \def\bbl@elt##1##2##3{%
2704 \def\bbl@section{##1}%
2705 \in@{=date.}{=##1}% Find a better place
2706 \ifin@
2707 \bbl@ifunset{bbl@inikv@##1}%
2708 {\bbl@ini@calendar{##1}}%
2709 {}%
2710 \fi
2711 \bbl@ifunset{bbl@inikv@##1}{}%
2712 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2713 \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.

```

2714 \def\bbl@extend@ini@aux#1{%
2715 \bbl@startcommands*{#1}{captions}%
2716 % Activate captions/... and modify exports
2717 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2718 \setlocalecaption{#1}{##1}{##2}}%
2719 \def\bbl@inikv@captions##1##2{%
2720 \bbl@ini@captions@aux{##1}{##2}}%
2721 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2722 \def\bbl@exportkey##1##2##3{%
2723 \bbl@ifunset{bbl@kv@##2}{}%
2724 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2725 \bbl@exp{\global\let<bbl@##1\language>\<bbl@kv@##2>}}%
2726 \fi}}%
2727 % As with \bbl@read@ini, but with some changes
2728 \bbl@read@ini@aux
2729 \bbl@ini@exports\tw@
2730 % Update inidata@lang by pretending the ini is read.
2731 \def\bbl@elt##1##2##3{%
2732 \def\bbl@section{##1}%
2733 \bbl@iniline##2=##3\bbl@iniline}%
2734 \csname bbl@inidata@#1\endcsname
2735 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2736 \StartBabelCommands*{#1}{date}% And from the import stuff
2737 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2738 \bbl@savetoday
2739 \bbl@savestate

```



```
2740 \bbl@endcommands}
```

A somewhat hackish tool to handle calendar sections.

```
2741 \def\bbl@ini@calendar#1{%
2742 \lowercase{\def\bbl@tempa{=#1=}}%
2743 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2744 \bbl@replace\bbl@tempa{=date.}{}%
2745 \in@{.licr=}{#1=}%
2746 \ifin@
2747 \ifcase\bbl@engine
2748 \bbl@replace\bbl@tempa{.licr=}{}%
2749 \else
2750 \let\bbl@tempa\relax
2751 \fi
2752 \fi
2753 \ifx\bbl@tempa\relax\else
2754 \bbl@replace\bbl@tempa{=}{}%
2755 \ifx\bbl@tempa\empty\else
2756 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2757 \fi
2758 \bbl@exp{%
2759 \def\<bbl@inikv@#1>####1####2{%
2760 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2761 \fi}
```

A key with a slash in \babelprovide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in \bbl@inistore above).

```
2762 \def\bbl@renewinikey#1/#2\@#3{%
2763 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2764 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2765 \bbl@trim\toks@{#3}% value
2766 \bbl@exp{%
2767 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2768 \\g@addto@macro\\bbl@inidata{%
2769 \\\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%}
```

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

```
2770 \def\bbl@exportkey#1#2#3{%
2771 \bbl@ifunset{\bbl@kv@#2}%
2772 {\bbl@csarg\gdef{#1@\language}\@empty}%
2773 {\expandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2774 \bbl@csarg\gdef{#1@\language}\@empty}%
2775 \else
2776 \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
2777 \fi}}
```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@ini@exports is called always (via \bbl@inise), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

Although BCP 47 doesn't treat 'x-' as an extension, the CLDR and many other sources do (as a *private use extension*). For consistency with other single-letter subtags or 'singletons', here is considered an extension, too.

The identification section is used internally by babel in the following places [to be completed]: BCP 47 script tag in the Unicode ranges, which is in turn used by onchar; the language system is set with the names, and then fontspec maps them to the opentype tags, but if the latter package doesn't define them, then babel does it; encodings are used in pdftex to select a font encoding valid (and preloaded) for a language loaded on the fly.

```
2778 \def\bbl@iniwarning#1{%
2779 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2780 {\bbl@warning%
```

```

2781      From babel-\bbl@cs{lini@\languagename}.ini:\\%
2782      \bbl@cs{@kv@identification.warning#1}\\%
2783      Reported}}}
2784 %
2785 \let\bbl@release@transforms\@empty
2786 \let\bbl@release@casing\@empty

```

Relevant keys are ‘exported’, i.e., global macros with short names are created with values taken from the corresponding keys. The number of exported keys depends on the loading level (#1): –1 and 0 only info (the identification section), 1 also basic (like linebreaking or character ranges), 2 also (re)new (with date and captions).

```

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

```

```

2839      \bbl@tglobal\bbl@savetoday
2840      \bbl@tglobal\bbl@savestate
2841      \bbl@savestrings
2842      \fi
2843      \fi}

```

4.20. Processing keys in ini

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

```

2844 \def\bbl@inikv#1#2{%      key=value
2845   \toks@{#2}%              This hides #'s from ini values
2846   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2847 \let\bbl@inikv@identification\bbl@inikv
2848 \let\bbl@inikv@date\bbl@inikv
2849 \let\bbl@inikv@typography\bbl@inikv
2850 \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.

```

2851 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\empty x-\fi}
2852 \def\bbl@inikv@characters#1#2{%
2853   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2854   {\bbl@exp{%
2855     \\g@addto@macro\\bbl@release@casing{%
2856       \\bbl@casemapping}{\language}\unexpanded{#2}}}%
2857   {\in@{casing.}{#1}% e.g., casing.Uv = uV
2858     \ifin@
2859       \lowercase{\def\bbl@tempb{#1}}%
2860       \bbl@replace\bbl@tempb{casing.}{}%
2861       \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2862         \\bbl@casemapping
2863         {\\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2864       \else
2865         \bbl@inikv{#1}{#2}%
2866       \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’.

```

2867 \def\bbl@inikv@counters#1#2{%
2868   \bbl@ifsamestring{#1}{digits}%
2869   {\bbl@error{digits-is-reserved}}{}{}%
2870   {}%
2871   \def\bbl@tempc{#1}%
2872   \bbl@trim@def{\bbl@tempb*}{#2}%
2873   \in@{.1$}{#1$}%
2874   \ifin@
2875     \bbl@replace\bbl@tempc{.1}{}%
2876     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
2877       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2878     \fi
2879     \in@{.F.}{#1}%
2880     \ifin@else\in@{.S.}{#1}\fi
2881     \ifin@
2882       \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
2883       \else
2884         \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2885         \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
2886         \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
2887         \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.

```
2888 \ifcase\bbl@engine
2889 \bbl@csarg\def{inikv@captions.licr}#1#2{%
2890 \bbl@ini@captions@aux{#1}{#2}}
2891 \else
2892 \def\bbl@inikv@captions#1#2{%
2893 \bbl@ini@captions@aux{#1}{#2}}
2894 \fi
```

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

```
2895 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2896 \bbl@replace\bbl@tempa{.template}{}}%
2897 \def\bbl@toreplace{#1}{}%
2898 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
2899 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
2900 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2901 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname{}}%
2902 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
2903 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2904 \ifin@
2905 \@nameuse{\bbl@patch\bbl@tempa}%
2906 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2907 \fi
2908 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2909 \ifin@
2910 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2911 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2912 \\\bbl@ifunset{\bbl@bbl@tempa fmt@\\\language}%
2913 {[fnum@\bbl@tempa]}%
2914 {\\\@nameuse{\bbl@bbl@tempa fmt@\\\language}}}}%
2915 \fi}
2916 %
2917 \def\bbl@ini@captions@aux#1#2{%
2918 \bbl@trim\def\bbl@tempa{#1}%
2919 \bbl@xin@{.template}{\bbl@tempa}%
2920 \ifin@
2921 \bbl@ini@captions@template{#2}\language
2922 \else
2923 \bbl@ifblank{#2}%
2924 {\bbl@exp{%
2925 \toks@{\\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2926 {\bbl@trim\toks@{#2}}}%
2927 \bbl@exp{%
2928 \\\bbl@add\\bbl@savestrings{%
2929 \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
2930 \toks@\expandafter{\bbl@captionslist}%
2931 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
2932 \ifin@else
2933 \bbl@exp{%
2934 \\\bbl@add\<\bbl@extracaps@\language>{\<\bbl@tempa name>}}%
2935 \\\bbl@tglobal\<\bbl@extracaps@\language>}}%
2936 \fi
2937 \fi}
```

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

```
2938 \def\bbl@list@the{%
2939 part,chapter,section,subsection,subsubsection,paragraph,%
2940 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2941 table,page,footnote,mpfootnote,mpfn}
2942 %
2943 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
```

```

2944 \bbl@ifunset{bbl@map@#1@\language}%
2945 {\@nameuse{#1}}%
2946 {\@nameuse{bbl@map@#1@\language}}}%
2947 %
2948 \def\bbl@map@lbl#1{% #1:a sign, eg, .
2949 \bbl@ifunset{bbl@map@#1@\language}%
2950 {#1}%
2951 {\@nameuse{bbl@map@#1@\language}}}%
2952 %
2953 \def\bbl@inikv@labels#1#2{%
2954 \in@{,dot.map,}{, #1,}%
2955 \ifin@
2956 \global\@namedef{bbl@map@.@\language}{#2}%
2957 \bbl@foreach\bbl@list@the{%
2958 \bbl@ifunset{the##1}{}%
2959 {\bbl@ncarg\let\bbl@tempd{the##1}%
2960 \bbl@carg\bbl@sreplace{the##1}{.}{\bbl@map@lbl{.}}%
2961 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2962 \bbl@exp{\gdef<the##1>{\[the##1]}}}%
2963 \fi}}}%
2964 \edef\bbl@tempb{enumi,enumii,enumiii,enumiv}%
2965 \bbl@foreach\bbl@tempb{%
2966 \bbl@ifunset{label##1}{}%
2967 {\bbl@ncarg\let\bbl@tempd{label##1}%
2968 \bbl@carg\bbl@sreplace{label##1}{.}{\bbl@map@lbl{.}}%
2969 \expandafter\ifx\csname label##1\endcsname\bbl@tempd\else
2970 \bbl@exp{\gdef<label##1>{\[label##1]}}}%
2971 \fi}}}%
2972 \else
2973 \in@{.map}{#1}%
2974 \ifin@
2975 \ifx\bbl@KVP@labels\@nnil\else
2976 \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
2977 \ifin@
2978 \def\bbl@tempc{#1}%
2979 \bbl@replace\bbl@tempc{.map}{}%
2980 \in@{, #2, }{, arabic, roman, Roman, alph, Alph, fnsymbol,}%
2981 \bbl@exp{%
2982 \gdef<bbl@map@bbl@tempc @\language>%
2983 {\ifin@<#2>\else\\localecounter{#2}\fi}}%
2984 \bbl@foreach\bbl@list@the{%
2985 \bbl@ifunset{the##1}{}%
2986 {\bbl@ncarg\let\bbl@tempd{the##1}%
2987 \bbl@exp{%
2988 \\bbl@sreplace<the##1>%
2989 {\<\bbl@tempc>{##1}}%
2990 {\bbl@map@cnt{\bbl@tempc}{##1}}%
2991 \\bbl@sreplace<the##1>%
2992 {\<\@empty @\bbl@tempc>\<c##1>}%
2993 {\bbl@map@cnt{\bbl@tempc}{##1}}%
2994 \\bbl@sreplace<the##1>%
2995 {\csname @\bbl@tempc\endcsname\<c##1>}%
2996 {\bbl@map@cnt{\bbl@tempc}{##1}}}%
2997 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2998 \bbl@exp{\gdef<the##1>{\[the##1]}}%
2999 \fi}}%
3000 \fi
3001 \fi
3002 %
3003 \else
3004 % The following code is still under study. You can test it and make
3005 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3006 % language dependent.

```

```

3007 \in@{enumerate.}{#1}%
3008 \ifin@
3009 \def\bbl@tempa{#1}%
3010 \bbl@replace\bbl@tempa{enumerate.}{}%
3011 \def\bbl@toreplace{#2}%
3012 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3013 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3014 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3015 \toks@{\expandafter\bbl@toreplace}%
3016 \bbl@exp{%
3017 \\\bbl@add<extras\language>{%
3018 \\\babel@save<labelenum\romannumeral\bbl@tempa>%
3019 \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3020 \\\bbl@tglobal<extras\language>}%
3021 \fi
3022 \fi
3023 \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.

```

3024 \def\bbl@chapttype{chapter}
3025 \ifx\@makechapterhead\undefined
3026 \let\bbl@patchchapter\relax
3027 \else\ifx\thechapter\undefined
3028 \let\bbl@patchchapter\relax
3029 \else\ifx\ps@headings\undefined
3030 \let\bbl@patchchapter\relax
3031 \else
3032 \def\bbl@patchchapter{%
3033 \global\let\bbl@patchchapter\relax
3034 \gdef\bbl@chfmt{%
3035 \bbl@ifunset{bbl@\bbl@chapttype fmt@\language}%
3036 {\@chapapp\space\thechapter}%
3037 {\@nameuse{bbl@\bbl@chapttype fmt@\language}}}%
3038 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3039 \bbl@sreplace\ps@headings{\@chapapp \thechapter}{\bbl@chfmt}%
3040 \bbl@sreplace\chaptermark{\@chapapp \thechapter}{\bbl@chfmt}%
3041 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3042 \bbl@tglobal\appendix
3043 \bbl@tglobal\ps@headings
3044 \bbl@tglobal\chaptermark
3045 \bbl@tglobal\@makechapterhead}
3046 \let\bbl@patchappendix\bbl@patchchapter
3047 \fi\fi\fi
3048 \ifx\@part\undefined
3049 \let\bbl@patchpart\relax
3050 \else
3051 \def\bbl@patchpart{%
3052 \global\let\bbl@patchpart\relax
3053 \gdef\bbl@partformat{%
3054 \bbl@ifunset{bbl@partfmt@\language}%
3055 {\partname\nobreakspace\thepart}%
3056 {\@nameuse{bbl@partfmt@\language}}}%
3057 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3058 \bbl@tglobal\@part}
3059 \fi

```

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

```

3060 \let\bbl@calendar\@empty
3061 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3062 \def\bbl@localedate#1#2#3#4{%

```

```

3063 \begingroup
3064 \edef\bbl@they{#2}%
3065 \edef\bbl@them{#3}%
3066 \edef\bbl@thed{#4}%
3067 \edef\bbl@tempe{%
3068     \bbl@ifunset{\bbl@calpr@\language\name}{\bbl@cl{calpr}},%
3069     #1}%
3070 \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3071 \bbl@replace\bbl@tempe{ }{}%
3072 \bbl@replace\bbl@tempe{convert}{convert=}%
3073 \let\bbl@ld@calendar\@empty
3074 \let\bbl@ld@variant\@empty
3075 \let\bbl@ld@convert\relax
3076 \def\bbl@tempb##1=##2\@{\@{\namedef{\bbl@ld@##1}{##2}}}%
3077 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3078 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3079 \ifx\bbl@ld@calendar\@empty\else
3080     \ifx\bbl@ld@convert\relax\else
3081         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3082         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3083     \fi
3084 \fi
3085 \@nameuse{\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3086 \edef\bbl@calendar{% Used in \month..., too
3087     \bbl@ld@calendar
3088     \ifx\bbl@ld@variant\@empty\else
3089         .\bbl@ld@variant
3090     \fi}%
3091 \bbl@cased
3092     {\@nameuse{\bbl@date@\language\name @\bbl@calendar}%
3093     \bbl@they\bbl@them\bbl@thed}%
3094 \endgroup}
3095 %
3096 \def\bbl@printdate#1{%
3097     \@ifnextchar{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}}
3098 \def\bbl@printdate@i#1[#2]#3#4#5{%
3099     \bbl@usedategrouptrue
3100     \@nameuse{\bbl@ensure@#1}{\localedate[#2][#3][#4][#5]}
3101 %
3102 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3103 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3104     \bbl@trim@def\bbl@tempa{#1.#2}%
3105     \bbl@ifsamestring{\bbl@tempa}{months.wide}%           to savedate
3106     {\bbl@trim@def\bbl@tempa{#3}%
3107     \bbl@trim\toks@{#5}%
3108     \@temptokena\expandafter{\bbl@savedate}%
3109     \bbl@exp{% Reverse order - in ini last wins
3110         \def\\bbl@savedate{%
3111             \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3112             \the\@temptokena}}}%
3113     {\bbl@ifsamestring{\bbl@tempa}{date.long}%           defined now
3114     {\lowercase{\def\bbl@tempb{#6}}}%
3115     \bbl@trim@def\bbl@toreplace{#5}%
3116     \bbl@TG@@date
3117     \global\bbl@csarg\let{date@\language\name @\bbl@tempb}\bbl@toreplace
3118     \ifx\bbl@savetoday\@empty
3119         \bbl@exp{%
3120             \\AfterBabelCommands{%
3121                 \gdef\<\language\name date>{\protect\<\language\name date >}%
3122                 \gdef\<\language\name date >{\bbl@printdate{\language\name}}}%
3123             \def\\bbl@savetoday{%
3124                 \\SetString\\today{%
3125                     \<\language\name date>[convert]}%

```

```

3126             {\the\year}{\the\month}{\the\day}}}%
3127     \fi}%
3128     {}}}

```

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).

```

3129 \let\bbl@calendar@empty
3130 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3131   \@nameuse{bbl@ca@#2}#1\@@}
3132 \newcommand\BabelDateSpace{\nobreakspace}
3133 \newcommand\BabelDateDot{.\@}
3134 \newcommand\BabelDated[1]{\number#1}
3135 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3136 \newcommand\BabelDateM[1]{\number#1}
3137 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3138 \newcommand\BabelDateMMMM[1]{%
3139   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3140 \newcommand\BabelDatey[1]{\number#1}%
3141 \newcommand\BabelDateyy[1]{%
3142   \ifnum#1<10 0\number#1 %
3143   \else\ifnum#1<100 \number#1 %
3144   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3145   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3146   \else
3147     \bbl@error{limit-two-digits}{\number#1}%
3148     \fi\fi\fi\fi}
3149 \newcommand\BabelDateyyy[1]{\number#1}
3150 \newcommand\BabelDateU[1]{\number#1}%
3151 \def\bbl@replace@finish@iii#1{%
3152   \bbl@exp{\def\#1###1####2####3{\the\toks@}}%
3153 \def\bbl@TG@@date{%
3154   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}}%
3155   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot}}%
3156   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3157   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3158   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3159   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3160   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{###2}}%
3161   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3162   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3163   \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3164   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3165   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[###1]}%
3166   \bbl@replace\bbl@toreplace{[U]}{\bbl@datecctr[###1]}%
3167   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[###2]}%
3168   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[###3]}%
3169   \bbl@replace@finish@iii\bbl@toreplace}
3170 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3171 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

4.21. French spacing (again)

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

```

3172 \AddToHook{begindocument/before}{%
3173   \let\bbl@normalsf\normalsfcodes
3174   \let\normalsfcodes\relax}
3175 \AtBeginDocument{%
3176   \ifx\bbl@normalsf\empty
3177     \ifnum\sfcodes\.=\@m

```



```

3178     \let\normalsfcodes\frenchspacing
3179   \else
3180     \let\normalsfcodes\nonfrenchspacing
3181   \fi
3182 \else
3183   \let\normalsfcodes\bbl@normalsf
3184 \fi}

```

Transforms.

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

```

3185 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3186 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3187 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3188   #1[#2]{#3}{#4}{#5}}
3189 \begingroup
3190   \catcode`\%=12
3191   \catcode`\&=14
3192   \gdef\bbl@transforms#1#2#3{%&
3193     \directlua{
3194       local str = [==[#2]==]
3195       str = str:gsub('%.%d+%.%d+$', '')
3196       token.set_macro('babeltempa', str)
3197     }&%
3198     \def\babeltempc{}&%
3199     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3200     \ifin@ \else
3201       \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&%
3202     \fi
3203     \ifin@
3204       \bbl@foreach\bbl@KVP@transforms{%&
3205         \bbl@xin@{: \babeltempa,}{,##1,}&%
3206         \ifin@ &% font:font:transform syntax
3207           \directlua{
3208             local t = {}
3209             for m in string.gmatch('##1'..' ':'(.)') do
3210               table.insert(t, m)
3211             end
3212             table.remove(t)
3213             token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3214           }&%
3215         \fi}&%
3216       \in@{.0$}{#2$}&%
3217       \ifin@
3218         \directlua{%& (\attribute) syntax
3219           local str = string.match([[ \bbl@KVP@transforms]],
3220             '%(([^%([-)%][^%)]-\babeltempa)')
3221           if str == nil then
3222             token.set_macro('babeltempb', '')
3223           else
3224             token.set_macro('babeltempb', ',attribute=' .. str)
3225           end
3226         }&%
3227       \toks@{#3}&%
3228       \bbl@exp{%&
3229         \\g@addto@macro\\ \bbl@release@transforms{%&
3230           \relax &% Closes previous \bbl@transforms@aux
3231           \\ \bbl@transforms@aux
3232             \\\#1{label=\babeltempa\babeltempb\babeltempc}&%
3233             {\language}\the\toks@}}&%

```

```

3234     \else
3235         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3236     \fi
3237 \fi}
3238 \endgroup

```

4.22. Handle language system

The language system (i.e., Language and Script) to be used when defining a font or setting the direction are set with the following macros. It also deals with unhyphenated line breaking in xetex (e.g., Thai and traditional Sanskrit), which is done with a hack at the font level because this engine doesn't support it.

```

3239 \def\bbl@provide@lsys#1{%
3240     \bbl@ifunset\bbl@lname@#1{%
3241         {\bbl@load@info{#1}}%
3242     }%
3243     \bbl@csarg\let{lsys@#1}\@empty
3244     \bbl@ifunset\bbl@sname@#1{\bbl@csarg\gdef{sname@#1}{Default}}{%
3245     \bbl@ifunset\bbl@sotf@#1{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3246     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3247     \bbl@ifunset\bbl@lname@#1{%
3248         {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3249     \ifcase\bbl@engine\or\or
3250         \bbl@ifunset\bbl@prehc@#1{%
3251             {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3252             }%
3253             {\ifx\bbl@xenoxyph\undefined
3254                 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3255                 \ifx\AtBeginDocument\@notprerr
3256                     \expandafter\@secondoftwo % to execute right now
3257                 \fi
3258                 \AtBeginDocument{%
3259                     \bbl@patchfont{\bbl@xenoxyph}%
3260                     {\expandafter\select@language\expandafter{\language}}}%
3261             \fi}%
3262     \fi
3263     \bbl@csarg\bbl@toglobal{lsys@#1}}

```

4.23. Numerals

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

```

3264 \def\bbl@setdigits#1#2#3#4#5{%
3265     \bbl@exp{%
3266         \def<\language name digits>####1% i.e., \langdigits
3267         \<\bbl@digits@language name>####1\\\@nil}%
3268         \let<\bbl@cntr@digits@language name>\<\language name digits>%
3269         \def<\language name counter>####1% i.e., \langcounter
3270         \\\expandafter<\bbl@counter@language name>%
3271         \\\csname c@####1\endcsname}%
3272         \def<\bbl@counter@language name>####1% i.e., \bbl@counter@lang
3273         \\\expandafter<\bbl@digits@language name>%
3274         \\\number####1\\\@nil}}%
3275 \def\bbl@tempa#1#2#3#4#5{%
3276     \bbl@exp{% Wow, quite a lot of hashes! :-(
3277         \def<\bbl@digits@language name>#####1{%
3278             \\\ifx#####1\\\@nil % i.e., \bbl@digits@lang
3279             \\\else
3280                 \\\ifx0#####1#1%
3281                 \\\else\\\ifx1#####1#2%
3282                 \\\else\\\ifx2#####1#3%

```

[illegible]

```

3295 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={
3296   \ifx\\#1%                % \\ before, in case #1 is multiletter
3297     \bbl@exp{%
3298       \def\\bbl@tempa####1{%
3299         \<ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3300   \else
3301     \toks@\expandafter{\the\toks@\or #1}%
3302     \expandafter\bbl@buildifcase
3303   \fi}

```

```

3304 \newcommand\localenumerall[2]{\bbl@cs{cnt@#1@\language@}{#2}}
3305 \def\bbl@localcnt#1#2{\localenumerall{#2}{#1}}
3306 \newcommand\localecounter[2]{%
3307   \expandafter\bbl@localcnt#1%
3308   \expandafter{\number\csname c@#2\endcsname}{#1}}
3309 \def\bbl@alphnumerall#1#2{%
3310   \expandafter\bbl@alphnumerall@i\number#2 76543210@@{#1}}
3311 \def\bbl@alphnumerall@i#1#2#3#4#5#6#7#8@@@#9{%
3312   \ifcase\car#8\@nil\or      % Currently <10000, but prepared for bigger
3313     \bbl@alphnumerall@ii{#9}000000#1\or
3314     \bbl@alphnumerall@ii{#9}00000#1#2\or
3315     \bbl@alphnumerall@ii{#9}0000#1#2#3\or
3316     \bbl@alphnumerall@ii{#9}000#1#2#3#4\else
3317     \bbl@alphnum@invalid{>9999}%
3318   \fi}
3319 \def\bbl@alphnumerall@ii#1#2#3#4#5#6#7#8{%
3320   \bbl@ifunset{\bbl@cnt@#1.F.\number#5#6#7#8@\language@}%
3321   {\bbl@cs{cnt@#1.4@\language@}{#5}
3322     \bbl@cs{cnt@#1.3@\language@}{#6}
3323     \bbl@cs{cnt@#1.2@\language@}{#7}
3324     \bbl@cs{cnt@#1.1@\language@}{#8}
3325     \ifnum#6#7#8>\z@
3326       \bbl@ifunset{\bbl@cnt@#1.S.321@\language@}{}%
3327       {\bbl@cs{cnt@#1.S.321@\language@}{}%
3328       \fi}%
3329   {\bbl@cs{cnt@#1.F.\number#5#6#7#8@\language@}}}
3330 \def\bbl@alphnum@invalid#1{%
3331   \bbl@error{alphabetic-too-large}{#1}{}}

```

```
3332 \newcommand\BabelUppercaseMapping[3]{%
3333   \DeclareUppercaseMapping[\nameuse{bbl@casing@#1}]{#2}{#3}}
3334 \newcommand\BabelTitlecaseMapping[3]{%
```

```

3335 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3336 \newcommand\BabelLowercaseMapping[3]{%
3337 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

The parser for casing and casing.⟨variant⟩.
3338 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3339 \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3340 \else
3341 \def\bbl@uftocode#1{\expandafter\string#1}
3342 \fi
3343 \def\bbl@casemapping#1#2#3{% 1:variant
3344 \def\bbl@tempa##1 ##2{% Loop
3345 \bbl@casemapping@i{##1}%
3346 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3347 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3348 \def\bbl@tempe{0}% Mode (upper/lower...)
3349 \def\bbl@tempc{#3}% Casing list
3350 \expandafter\bbl@tempa\bbl@tempc\@empty}
3351 \def\bbl@casemapping@i#1{%
3352 \def\bbl@tempb{#1}%
3353 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3354 \@nameuse{regex_replace_all:nnN}%
3355 {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\0}}\bbl@tempb
3356 \else
3357 \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb
3358 \fi
3359 \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3360 \def\bbl@casemapping@ii#1#2#3\@@{%
3361 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3362 \ifin@
3363 \edef\bbl@tempe{%
3364 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3365 \else
3366 \ifcase\bbl@tempe\relax
3367 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3368 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3369 \or
3370 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3371 \or
3372 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3373 \or
3374 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3375 \fi
3376 \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.

```

3377 \def\bbl@localeinfo#1#2{%
3378 \bbl@ifunset{bbl@info@#2}{#1}%
3379 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\language}{#1}%
3380 {\bbl@cs{csname bbl@info@#2\endcsname @\language}}}}
3381 \newcommand\localeinfo[1]{%
3382 \ifx*#1\@empty
3383 \bbl@afterelse\bbl@localeinfo{}%
3384 \else
3385 \bbl@localeinfo
3386 {\bbl@error{no-ini-info}}{}{}{}%
3387 {#1}%
3388 \fi}
3389 % \@namedef{bbl@info@name.locale}{lcname}
3390 \@namedef{bbl@info@tag.ini}{lini}
3391 \@namedef{bbl@info@name.english}{elname}

```

```

3392 \@namedef{bbl@info@name.opentype}{\lname}
3393 \@namedef{bbl@info@tag.bcp47}{\tbc}
3394 \@namedef{bbl@info@language.tag.bcp47}{\lbc}
3395 \@namedef{bbl@info@tag.opentype}{\lotf}
3396 \@namedef{bbl@info@script.name}{\esname}
3397 \@namedef{bbl@info@script.name.opentype}{\sname}
3398 \@namedef{bbl@info@script.tag.bcp47}{\sbcp}
3399 \@namedef{bbl@info@script.tag.opentype}{\sotf}
3400 \@namedef{bbl@info@region.tag.bcp47}{\rbcp}
3401 \@namedef{bbl@info@variant.tag.bcp47}{\vbcp}
3402 \@namedef{bbl@info@extension.t.tag.bcp47}{\extt}
3403 \@namedef{bbl@info@extension.u.tag.bcp47}{\extu}
3404 \@namedef{bbl@info@extension.x.tag.bcp47}{\extx}

```

With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it. Since the info in ini files are always loaded, it has been made no-op in version 25.8.

```

3405 <<{*More package options}>> ≡
3406 \DeclareOption{ensureinfo=off}{}
3407 <</More package options>>
3408 \let\BabelEnsureInfo\relax

```

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

```

3409 \newcommand\getlocaleproperty{%
3410   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3411 \def\bbl@getproperty@s#1#2#3{%
3412   \let#1\relax
3413   \def\bbl@elt##1##2##3{%
3414     \bbl@ifsamestring{##1/##2}{#3}%
3415     {\providecommand#1{##3}%
3416     \def\bbl@elt###1###2###3{}}}%
3417   {}}%
3418   \bbl@cs{inidata@#2}}%
3419 \def\bbl@getproperty@x#1#2#3{%
3420   \bbl@getproperty@s{#1}{#2}{#3}%
3421   \ifx#1\relax
3422     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3423   \fi}

```

To inspect every possible loaded ini, we define \LocaleForEach, where \bbl@ini@loaded is a comma-separated list of locales, built by \bbl@read@ini.

```

3424 \let\bbl@ini@loaded\@empty
3425 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3426 \def\ShowLocaleProperties#1{%
3427   \typeout{}%
3428   \typeout{*** Properties for language '#1' ***}
3429   \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3430   \@nameuse{bbl@inidata@#1}%
3431   \typeout{*****}}

```

4.26. BCP 47 related commands

This macro is called by language selectors when the language isn't recognized. So, it's the core for (1) mapping from a BCP 47 tag to the actual language, if bcp47.toname is enabled (i.e., if bbl@bcptoname is true), and (2) lazy loading. With autoload.bcp47 enabled *and* lazy loading, we must first build a name for the language, with the help of autoload.bcp47.prefix. Then we use \provideprovide passing the options set with autoload.bcp47.options (by default import). Finally, and if the locale has not been loaded before, we use \provideprovide with the language name as passed to the selector.

```

3432 \newif\ifbbl@bcpallowed
3433 \bbl@bcpallowedfalse
3434 \def\bbl@autoload@options{@import}
3435 \def\bbl@provide@locale{%
3436   \ifx\babelprovide\undefined

```

```

3437 \bbl@error{base-on-the-fly}{\}{\}%
3438 \fi
3439 \let\bbl@auxname\language
3440 \ifbbl@bcptoname
3441 \bbl@ifunset{\bbl@bcp@map@\language}{\}% Move uplevel??
3442 {\edef\language{\@nameuse{\bbl@bcp@map@\language}}}%
3443 \let\localname\language}%
3444 \fi
3445 \ifbbl@bcpallowed
3446 \expandafter\ifx\csname date\language\endcsname\relax
3447 \expandafter
3448 \bbl@bcplookup\language-\@empty-\@empty-\@empty@@
3449 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3450 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3451 \let\localname\language
3452 \expandafter\ifx\csname date\language\endcsname\relax
3453 \let\bbl@initoload\bbl@bcp
3454 \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3455 \let\bbl@initoload\relax
3456 \fi
3457 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localname}%
3458 \fi
3459 \fi
3460 \fi
3461 \expandafter\ifx\csname date\language\endcsname\relax
3462 \IfFileExists{babel-\language.tex}%
3463 {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3464 {\}%
3465 \fi}

```

\TeX needs to know the BCP 47 codes for some features. For that, it expects `\BCPdata` to be defined. While language, region, script, and variant are recognized, extension `\langle s \rangle` for singletons may change.

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

```

3466 \providecommand\BCPdata{}
3467 \ifx\renewcommand\undefined\else
3468 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3469 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3470 \@nameuse{\str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3471 {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3472 {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3473 \def\bbl@bcpdata@ii#1#2{%
3474 \bbl@ifunset{\bbl@info@#1.tag.bcp47}%
3475 {\bbl@error{unknown-ini-field}{#1}{\}}}%
3476 {\bbl@ifunset{\bbl@csname\bbl@info@#1.tag.bcp47\endcsname @#2}{\}%
3477 {\bbl@cs{\csname\bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3478 \fi
3479 \@namedef{\bbl@info@casing.tag.bcp47}{casing}
3480 \@namedef{\bbl@info@tag.tag.bcp47}{tbc} % For \BCPdata

```

5. Adjusting the Babel behavior

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

```

3481 \newcommand\babeladjust[1]{%
3482 \bbl@forkv{#1}{%
3483 \bbl@ifunset{\bbl@ADJ@##1@##2}%
3484 {\bbl@cs{ADJ@##1}{##2}}%
3485 {\bbl@cs{ADJ@##1@##2}}}%
3486 %
3487 \def\bbl@adjust@lua#1#2{%
3488 \ifvmode

```

```

3489 \ifnum\currentgrouplevel=\z@
3490 \directlua{ Babel.#2 }%
3491 \expandafter\expandafter\expandafter\@gobble
3492 \fi
3493 \fi
3494 {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3495 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3496 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3497 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3498 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3499 \@namedef{bbl@ADJ@bidi.text@on}{%
3500 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3501 \@namedef{bbl@ADJ@bidi.text@off}{%
3502 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3503 \@namedef{bbl@ADJ@bidi.math@on}{%
3504 \let\bbl@noamsmath\@empty}
3505 \@namedef{bbl@ADJ@bidi.math@off}{%
3506 \let\bbl@noamsmath\relax}
3507 %
3508 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3509 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3510 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3511 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3512 %
3513 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3514 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3515 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3516 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3517 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3518 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3519 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3520 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3521 \@namedef{bbl@ADJ@justify.arabic@on}{%
3522 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3523 \@namedef{bbl@ADJ@justify.arabic@off}{%
3524 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3525 %
3526 \def\bbl@adjust@layout#1{%
3527 \ifvmode
3528 #1%
3529 \expandafter\@gobble
3530 \fi
3531 {\bbl@error{layout-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3532 \@namedef{bbl@ADJ@layout.tabular@on}{%
3533 \ifnum\bbl@tabular@mode=\tw@
3534 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3535 \else
3536 \chardef\bbl@tabular@mode\@ne
3537 \fi}
3538 \@namedef{bbl@ADJ@layout.tabular@off}{%
3539 \ifnum\bbl@tabular@mode=\tw@
3540 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3541 \else
3542 \chardef\bbl@tabular@mode\z@
3543 \fi}
3544 \@namedef{bbl@ADJ@layout.lists@on}{%
3545 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3546 \@namedef{bbl@ADJ@layout.lists@off}{%
3547 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3548 %
3549 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3550 \bbl@bcpallowedtrue}
3551 \@namedef{bbl@ADJ@autoload.bcp47@off}{%

```

```

3552 \bbl@bcpallowedfalse}
3553 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3554 \def\bbl@bcp@prefix{#1}}
3555 \def\bbl@bcp@prefix{bcp47-}
3556 \@namedef{bbl@ADJ@autoload.options}#1{%
3557 \def\bbl@autoload@options{#1}}
3558 \def\bbl@autoload@bcptoptions{import}
3559 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3560 \def\bbl@autoload@bcptoptions{#1}}
3561 \newif\ifbbl@bcptname
3562 %
3563 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3564 \bbl@bcptnametrue}
3565 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3566 \bbl@bcptnamefalse}
3567 %
3568 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3569 \directlua{ Babel.ignore_pre_char = function(node)
3570 return (node.lang == \the\csname l@nohyphenation\endcsname)
3571 end }}
3572 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3573 \directlua{ Babel.ignore_pre_char = function(node)
3574 return false
3575 end }}
3576 %
3577 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3578 \def\bbl@ignoreinterchar{%
3579 \ifnum\language=\l@nohyphenation
3580 \expandafter\@gobble
3581 \else
3582 \expandafter\@firstofone
3583 \fi}}
3584 \@namedef{bbl@ADJ@interchar.disable@off}{%
3585 \let\bbl@ignoreinterchar\@firstofone}
3586 %
3587 \@namedef{bbl@ADJ@select.write@shift}{%
3588 \let\bbl@restorelastskip\relax
3589 \def\bbl@savelastskip{%
3590 \let\bbl@restorelastskip\relax
3591 \ifvmode
3592 \ifdim\lastskip=\z@
3593 \let\bbl@restorelastskip\nobreak
3594 \else
3595 \bbl@exp{%
3596 \def\\bbl@restorelastskip{%
3597 \skip@=\the\lastskip
3598 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3599 \fi
3600 \fi}}
3601 \@namedef{bbl@ADJ@select.write@keep}{%
3602 \let\bbl@restorelastskip\relax
3603 \let\bbl@savelastskip\relax}
3604 \@namedef{bbl@ADJ@select.write@omit}{%
3605 \AddBabelHook{babel-select}{beforestart}{%
3606 \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3607 \let\bbl@restorelastskip\relax
3608 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3609 \@namedef{bbl@ADJ@select.encoding@off}{%
3610 \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.

```
3611 <<*More package options>> ≡
3612 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3613 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3614 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3615 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3616 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3617 <</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.

```
3618 \bbl@trace{Cross referencing macros}
3619 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3620   \def\@newl@bel#1#2#3{%
3621     {\@safe@activetrue
3622       \bbl@ifunset{#1@#2}%
3623       \relax
3624       {\gdef\@multiplelabels{%
3625         \@latex@warning@no@line{There were multiply-defined labels}}%
3626         \@latex@warning@no@line{Label `#2' multiply defined}}%
3627       \global\@namedef{#1@#2}{#3}}}
```

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

```
3628 \CheckCommand*\@testdef[3]{%
3629   \def\reserved@a{#3}%
3630   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3631   \else
3632     \@tempswatrue
3633   \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.

```
3634 \def\@testdef#1#2#3{%
3635   \@safe@activetrue
3636   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3637   \def\bbl@tempb{#3}%
3638   \@safe@activesfalse
3639   \ifx\bbl@tempa\relax
3640   \else
3641     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3642   \fi
3643   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3644   \ifx\bbl@tempa\bbl@tempb
3645   \else
3646     \@tempswatrue
3647   \fi}
3648 \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.

```

3649 \bbl@xin@{R}\bbl@opt@safe
3650 \ifin@
3651 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3652 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3653 {\expandafter\strip@prefix\meaning\ref}%
3654 \ifin@
3655 \bbl@redefine\@kernel@ref#1{%
3656   \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetruefalse}
3657 \bbl@redefine\@kernel@pageref#1{%
3658   \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetruefalse}
3659 \bbl@redefine\@kernel@sref#1{%
3660   \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetruefalse}
3661 \bbl@redefine\@kernel@spageref#1{%
3662   \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetruefalse}
3663 \else
3664 \bbl@redefineroquest\ref#1{%
3665   \@safe@activetrue\org@ref{#1}\@safe@activetruefalse}
3666 \bbl@redefineroquest\pageref#1{%
3667   \@safe@activetrue\org@pageref{#1}\@safe@activetruefalse}
3668 \fi
3669 \else
3670 \let\org@ref\ref
3671 \let\org@pageref\pageref
3672 \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.

```

3673 \bbl@xin@{B}\bbl@opt@safe
3674 \ifin@
3675 \bbl@redefine\@citex[#1]#2{%
3676   \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetruefalse
3677   \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.)

```

3678 \AtBeginDocument{%
3679   \@ifpackageloaded{natbib}{%
3680     \def\@citex[#1][#2]#3{%
3681       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activetruefalse
3682       \org@@citex[#1][#2]{\bbl@tempa}}%
3683   }{}

```

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

```

3684 \AtBeginDocument{%
3685   \@ifpackageloaded{cite}{%
3686     \def\@citex[#1]#2{%
3687       \@safe@activetrue\org@@citex[#1][#2]\@safe@activetruefalse}%
3688   }{}

```

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

```
3689 \bbl@redefine\nocite#1{%
3690   \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3691 \bbl@redefine\bibcite{%
3692   \bbl@cite@choice
3693   \bibcite}
```

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

```
3694 \def\bbl@bibcite#1#2{%
3695   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3696 \def\bbl@cite@choice{%
3697   \global\let\bibcite\bbl@bibcite
3698   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3699   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3700   \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.

```
3701 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3702 \bbl@redefine\@bibitem#1{%
3703   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3704 \else
3705   \let\org@nocite\nocite
3706   \let\org@@citex\@citex
3707   \let\org@bibcite\bibcite
3708   \let\org@@bibitem\@bibitem
3709 \fi
```

5.2. Layout

```
3710 \newcommand\BabelPatchSection[1]{%
3711   \@ifundefined{#1}{}{%
3712     \bbl@exp{\let\bbl@ss@#1>\<#1}>%
3713     \@namedef{#1}{%
3714       \@ifstar{\bbl@presec@#1}{%
3715         {\@dblarg{\bbl@presec@x{#1}}}}
3716 \def\bbl@presec@x#1[#2]#3{%
3717   \bbl@exp{%
3718     \\select@language@x{\bbl@main@language}%
3719     \\bbl@cs{sspre@#1}%
3720     \\bbl@cs{ss@#1}%
3721     [\\foreignlanguage{\language@}{\unexpanded{#2}}}%
3722     {\\foreignlanguage{\language@}{\unexpanded{#3}}}%
3723     \\select@language@x{\language@}}}
```

```

3724 \def\bbl@presec@s#1#2{%
3725   \bbl@exp{%
3726     \select@language{x{\bbl@main@language}%
3727     \bbl@cs{sspre@#1}%
3728     \bbl@cs{ss@#1}*%
3729     {\foreignlanguage{\language}{\unexpanded{#2}}}%
3730     \select@language{x{\language}}}
3731 %
3732 \IfBabelLayout{sectioning}%
3733   {\BabelPatchSection{part}%
3734   \BabelPatchSection{chapter}%
3735   \BabelPatchSection{section}%
3736   \BabelPatchSection{subsection}%
3737   \BabelPatchSection{subsubsection}%
3738   \BabelPatchSection{paragraph}%
3739   \BabelPatchSection{subparagraph}%
3740   \def\babel@toc#1{%
3741     \select@language{x{\bbl@main@language}}}{%
3742 \IfBabelLayout{captions}%
3743   {\BabelPatchSection{caption}}}{%

```

\BabelFootnote Footnotes.

```

3744 \bbl@trace{Footnotes}
3745 \def\bbl@footnote#1#2#3{%
3746   \ifnextchar[%
3747     {\bbl@footnote@o{#1}{#2}{#3}}%
3748     {\bbl@footnote@x{#1}{#2}{#3}}}
3749 \long\def\bbl@footnote@x#1#2#3#4{%
3750   \bgroup
3751   \select@language{x{\bbl@main@language}%
3752   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3753   \egroup}
3754 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3755   \bgroup
3756   \select@language{x{\bbl@main@language}%
3757   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3758   \egroup}
3759 \def\bbl@footnotetext#1#2#3{%
3760   \ifnextchar[%
3761     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3762     {\bbl@footnotetext@x{#1}{#2}{#3}}}
3763 \long\def\bbl@footnotetext@x#1#2#3#4{%
3764   \bgroup
3765   \select@language{x{\bbl@main@language}%
3766   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3767   \egroup}
3768 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3769   \bgroup
3770   \select@language{x{\bbl@main@language}%
3771   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3772   \egroup}
3773 \def\BabelFootnote#1#2#3#4{%
3774   \ifx\bbl@fn@footnote\undefined
3775     \let\bbl@fn@footnote\footnote
3776   \fi
3777   \ifx\bbl@fn@footnotetext\undefined
3778     \let\bbl@fn@footnotetext\footnotetext
3779   \fi
3780   \bbl@iifblank{#2}%
3781     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3782     \namedef{\bbl@stripslash#1text}%
3783     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3784   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%

```

```

3785 \namedef{\bbl@stripslash#1text}%
3786 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}%
3787 \IfBabelLayout{footnotes}%
3788 {\let\bbl@OL@footnote\footnote
3789 \BabelFootnote\footnote\languagesname{}}%
3790 \BabelFootnote\localfootnote\languagesname{}}%
3791 \BabelFootnote\mainfootnote{}}{}%
3792 {}

```

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.

```

3793 \bbl@trace{Marks}
3794 \IfBabelLayout{sectioning}
3795 {\ifx\bbl@opt@headfoot\@nnil
3796 \g@addto@macro\resetactivechars{%
3797 \set@typeset@protect
3798 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3799 \let\protect\noexpand
3800 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3801 \edef\thepage{%
3802 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3803 \fi}%
3804 \fi}
3805 {\ifbbl@single\else
3806 \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3807 \markright#1}%
3808 \bbl@ifblank{#1}%
3809 {\org@markright{}}%
3810 {\toks@{#1}%
3811 \bbl@exp{%
3812 \org@markright{\protect\foreignlanguage{\languagesname}%
3813 \protect\bbl@restore@actives\the\toks@}}}%

```

\markboth

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

```

3814 \ifx\@mkboth\markboth
3815 \def\bbl@tempc{\let\@mkboth\markboth}%
3816 \else
3817 \def\bbl@tempc{}%
3818 \fi
3819 \bbl@ifunset{markboth }{\bbl@redefine\bbl@redefineroobust
3820 \markboth#1#2{%
3821 \protected@edef\bbl@tempb##1{%
3822 \protect\foreignlanguage
3823 {\languagesname}{\protect\bbl@restore@actives##1}}%
3824 \bbl@ifblank{#1}%
3825 {\toks@{}}%
3826 {\toks@\expandafter{\bbl@tempb{#1}}}%
3827 \bbl@ifblank{#2}%
3828 {\@temptokena{}}%
3829 {\@temptokena\expandafter{\bbl@tempb{#2}}}%

```

```

3830      \bbl@exp{\org@markboth{\the\toks@}{\the\temptokena}}}%
3831      \bbl@tempc
3832      \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.

```

3833 \bbl@trace{Preventing clashes with other packages}
3834 \ifx\org@ref\undefined\else
3835   \bbl@xin@{R}\bbl@opt@safe
3836   \ifin@
3837     \AtBeginDocument{%
3838       \@ifpackageloaded{ifthen}{%
3839         \bbl@redefine@long\ifthenelse#1#2#3{%
3840           \let\bbl@temp@pref\pageref
3841           \let\pageref\org@pageref
3842           \let\bbl@temp@ref\ref
3843           \let\ref\org@ref
3844           \@safe@activestrue
3845           \org@ifthenelse{#1}%
3846             {\let\pageref\bbl@temp@pref
3847              \let\ref\bbl@temp@ref
3848              \@safe@activesfalse
3849              #2}%
3850             {\let\pageref\bbl@temp@pref
3851              \let\ref\bbl@temp@ref
3852              \@safe@activesfalse
3853              #3}%
3854           }%
3855         }{}%
3856       }
3857 \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`.

```

3858 \AtBeginDocument{%
3859   \@ifpackageloaded{varioref}{%
3860     \bbl@redefine\@@vpageref#1[#2]#3{%
3861       \@safe@activestrue
3862       \org@@@vpageref{#1}[#2]{#3}%

```

```

3863      \@safe@activesfalse}%
3864      \bbl@redefine\vrefpagemum#1#2{%
3865      \@safe@activestrue
3866      \org\vrefpagemum{#1}{#2}%
3867      \@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.

```

3868      \expandafter\def\csname Ref \endcsname#1{%
3869      \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3870      }{}%
3871  }
3872 \fi

```

5.4.3. `hhline`

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the ‘:’ character which is made active by the french support in `babel`. Therefore we need to *reload* the package when the ‘:’ is an active character. Note that this happens *after* the category code of the @-sign has been changed to other, so we need to temporarily change it to letter again.

```

3873 \AtEndOfPackage{%
3874   \AtBeginDocument{%
3875     \@ifpackageloaded{hhline}%
3876     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3877     \else
3878       \makeatletter
3879       \def\@currname{hhline}\input{hhline.sty}\makeatother
3880       \fi}%
3881     {}}}

```

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

```

3882 \def\substitutefontfamily#1#2#3{%
3883   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3884   \immediate\write15{%
3885     \string\ProvidesFile{#1#2.fd}%
3886     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3887     \space generated font description file]^J
3888     \string\DeclareFontFamily{#1}{#2}{}^J
3889     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^J
3890     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^J
3891     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^J
3892     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^J
3893     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^J
3894     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^J
3895     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^J
3896     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^J
3897   }%
3898   \closeout15
3899 }
3900 \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

```

3901 \bbl@trace{Encoding and fonts}
3902 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3903 \newcommand\BabelNonText{TS1,T3,TS3}
3904 \let\org@TeX\TeX
3905 \let\org@LaTeX\LaTeX
3906 \let\ensureascii@firstofone
3907 \let\asciientcoding\@empty
3908 \AtBeginDocument{%
3909   \def\@elt#1{,#1,}%
3910   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3911   \let\@elt\relax
3912   \let\bbl@tempb\@empty
3913   \def\bbl@tempc{OT1}%
3914   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3915     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3916   \bbl@foreach\bbl@tempa{%
3917     \bbl@xin@{,#1}{,\BabelNonASCII,}%
3918     \ifin@
3919       \def\bbl@tempb{#1}% Store last non-ascii
3920     \else\bbl@xin@{,#1}{,\BabelNonText,}% Pass
3921     \ifin@else
3922       \def\bbl@tempc{#1}% Store last ascii
3923     \fi
3924   \fi}%
3925   \ifx\bbl@tempb\@empty\else
3926     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3927     \ifin@else
3928       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3929     \fi
3930   \let\asciientcoding\bbl@tempc
3931   \renewcommand\ensureascii[1]{%
3932     {\fontencoding{\asciientcoding}\selectfont#1}}%
3933   \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3934   \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3935   \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.

```

3936 \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.

```

3937 \AtBeginDocument{%
3938   \ifpackageloaded{fontspec}%
3939   {\xdef\latinencoding{%
3940     \ifx\UTFencname\undefined
3941       EU\ifcase\bbl@engine\or2\or1\fi
3942     \else
3943       \UTFencname
3944     \fi}}%
3945   {\gdef\latinencoding{OT1}%
3946     \ifx\cf@encoding\bbl@t@one
3947       \xdef\latinencoding{\bbl@t@one}%

```



```

3948 \else
3949 \def\elt#1{,#1,}%
3950 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3951 \let\elt\relax
3952 \bbl@xin@{,Tl,}\bbl@tempa
3953 \ifin@
3954 \xdef\latinencoding{\bbl@t@one}%
3955 \fi
3956 \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.

```

3957 \DeclareRobustCommand{\latintext}{%
3958 \fontencoding{\latinencoding}\selectfont
3959 \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.

```

3960 \ifx\@undefined\DeclareTextFontCommand
3961 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3962 \else
3963 \DeclareTextFontCommand{\textlatin}{\latintext}
3964 \fi

```

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

```

3965 \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.

```

3966 \bbl@trace{Loading basic (internal) bidi support}
3967 \ifodd\bbl@engine
3968 \else % Any xe+lua bidi
3969 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3970 \bbl@error{bidi-only-lua}{\}\}\}%
3971 \let\bbl@beforeforeign\leavevmode
3972 \AtEndOfPackage{%
3973 \EnableBabelHook{babel-bidi}%
3974 \bbl@xebidipar}
3975 \fi\fi
3976 \def\bbl@loadxebidi#1{%

```

```

3977 \ifx\RTLfootnotetext\@undefined
3978 \AtEndOfPackage{%
3979 \EnableBabelHook{babel-bidi}%
3980 \ifx\fontspec\@undefined
3981 \usepackage{fontspec}% bidi needs fontspec
3982 \fi
3983 \usepackage#1{bidi}%
3984 \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3985 \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3986 \ifnum\@nameuse{bbl@wdir@languagename}=\tw@ % 'AL' bidi
3987 \bbl@digitsdotdash % So ignore in 'R' bidi
3988 \fi}}%
3989 \fi}
3990 \ifnum\bbl@bidimode>200 % Any xe bidi=
3991 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3992 \bbl@tentative{bidi=bidi}
3993 \bbl@loadxebidi{}
3994 \or
3995 \bbl@loadxebidi{[rldocument]}
3996 \or
3997 \bbl@loadxebidi{}
3998 \fi
3999 \fi
4000 \fi
4001 \ifnum\bbl@bidimode=\@ne % bidi=default
4002 \let\bbl@beforeforeign\leavevmode
4003 \ifodd\bbl@engine % lua
4004 \newattribute\bbl@attr@dir
4005 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4006 \bbl@exp{\output{\bodydir\pagedir\the\output}}
4007 \fi
4008 \AtEndOfPackage{%
4009 \EnableBabelHook{babel-bidi}% pdf/lua/xe
4010 \ifodd\bbl@engine\else % pdf/xe
4011 \bbl@xebidipar
4012 \fi}
4013 \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.

```

4014 \bbl@trace{Macros to switch the text direction}
4015 \def\bbl@alscripts{%
4016 ,Arabic,Syriac,Thaana,Hanifi_Rohingya,Hanifi,Sogdian,}
4017 \def\bbl@rscripts{%
4018 Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4019 Hatran,Hebrew,Imperial_Aramaic,Inscriptional_Pahlavi,%
4020 Inscriptional_Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
4021 Mende_Kikakui,Meroitic_Cursive,Meroitic_Hieroglyphs,Nabataean,%
4022 Nko,Old_Hungarian,Old_North_Arabian,Old_Sogdian,%
4023 Old_South_Arabian,Old_Turkic,Old_Uyghur,Palmyrene,Phoenician,%
4024 Psalter_Pahlavi,Samaritan,Yezidi,Mandaean,%
4025 Meroitic,N'Ko,Orkhon,Todhri}
4026 %
4027 \def\bbl@provide@dirs#1{%
4028 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4029 \ifin@
4030 \global\bbl@csarg\chardef{wdir@#1}\@ne
4031 \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4032 \ifin@
4033 \global\bbl@csarg\chardef{wdir@#1}\tw@
4034 \fi
4035 \else

```

```

4036 \global\bbbl@csarg\chardef{wdir@#1}\z@
4037 \fi
4038 \ifodd\bbbl@engine
4039 \bbbl@csarg\ifcase{wdir@#1}%
4040 \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4041 \or
4042 \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4043 \or
4044 \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4045 \fi
4046 \fi}
4047 %
4048 \def\bbbl@switchdir{%
4049 \bbbl@ifunset{bbbl@sys{\languagename}}{\bbbl@provide@sys{\languagename}}{}%
4050 \bbbl@ifunset{bbbl@wdir{\languagename}}{\bbbl@provide@dirs{\languagename}}{}%
4051 \bbbl@exp{\bbbl@setdirs\bbbl@cl{wdir}}}%
4052 \def\bbbl@setdirs#1{%
4053 \ifcase\bbbl@select@type
4054 \bbbl@bodydir{#1}%
4055 \bbbl@pardir{#1}% <- Must precede \bbbl@texdir
4056 \fi
4057 \bbbl@texdir{#1}}
4058 \ifnum\bbbl@bidimode>\z@
4059 \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4060 \DisableBabelHook{babel-bidi}
4061 \fi

```

Now the engine-dependent macros.

```

4062 \ifodd\bbbl@engine % latex=1
4063 \else % pdftex=0, xetex=2
4064 \newcount\bbbl@dirlevel
4065 \chardef\bbbl@thetexdir\z@
4066 \chardef\bbbl@thepardir\z@
4067 \def\bbbl@texdir#1{%
4068 \ifcase#1\relax
4069 \chardef\bbbl@thetexdir\z@
4070 \@nameuse{setlatin}%
4071 \bbbl@texdir@i\beginL\endL
4072 \else
4073 \chardef\bbbl@thetexdir\@ne
4074 \@nameuse{setnonlatin}%
4075 \bbbl@texdir@i\beginR\endR
4076 \fi}
4077 \def\bbbl@texdir@i#1#2{%
4078 \ifhmode
4079 \ifnum\currentgrouplevel>\z@
4080 \ifnum\currentgrouplevel=\bbbl@dirlevel
4081 \bbbl@error{multiple-bidi}{}{}%
4082 \bgroup\aftergroup#2\aftergroup\egroup
4083 \else
4084 \ifcase\currentgrouptype\or % 0 bottom
4085 \aftergroup#2% 1 simple {}
4086 \or
4087 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4088 \or
4089 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4090 \or\or\or % vbox vtop align
4091 \or
4092 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4093 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4094 \or
4095 \aftergroup#2% 14 \begingroup
4096 \else

```

```

4097         \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4098     \fi
4099     \fi
4100     \bbl@dirlevel\currentgrouplevel
4101     \fi
4102     #1%
4103     \fi}
4104 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4105 \let\bbl@bodydir\@gobble
4106 \let\bbl@pagedir\@gobble
4107 \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).

```

4108 \def\bbl@xebidipar{%
4109     \let\bbl@xebidipar\relax
4110     \TeXeTstate\@ne
4111     \def\bbl@xeeverypar{%
4112         \ifcase\bbl@thepardir
4113             \ifcase\bbl@thetextdir\else\beginR\fi
4114         \else
4115             {\setbox\z@\lastbox\beginR\box\z@}%
4116         \fi}%
4117     \AddToHook{para/begin}{\bbl@xeeverypar}}
4118 \ifnum\bbl@bidimode>200 % Any xe bidi=
4119     \let\bbl@textdir\i\@gobbletwo
4120     \let\bbl@xebidipar\@empty
4121     \AddBabelHook{bidi}{foreign}{%
4122         \ifcase\bbl@thetextdir
4123             \BabelWrapText{\LR{##1}}%
4124         \else
4125             \BabelWrapText{\RL{##1}}%
4126         \fi}
4127     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4128 \fi
4129 \fi

```

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

```

4130 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}
4131 \AtBeginDocument{%
4132     \ifx\pdfstringdefDisableCommands\undefined\else
4133         \ifx\pdfstringdefDisableCommands\relax\else
4134             \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4135         \fi
4136     \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.

```

4137 \bbl@trace{Local Language Configuration}
4138 \ifx\loadlocalcfg\undefined
4139     \@ifpackagewith{babel}{noconfigs}%
4140     {\let\loadlocalcfg\@gobble}%
4141     {\def\loadlocalcfg#1{%
4142         \InputIfFileExists{#1.cfg}%
4143         {\typeout{*****^J%
4144             * Local config file #1.cfg used^^J%

```

```

4145          *}}}%
4146      \@empty}}
4147 \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).

```

4148 \bbl@trace{Language options}
4149 \def\BabelDefinitionFile#1#2#3{%
4150 \let\bbl@afterlang\relax
4151 \let\BabelModifiers\relax
4152 \let\bbl@loaded\@empty
4153 \def\bbl@load@language#1{%
4154   \InputIfFileExists{#1.ldf}%
4155   {\edef\bbl@loaded{\CurrentOption
4156     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4157   \expandafter\let\expandafter\bbl@afterlang
4158     \csname\CurrentOption.ldf-h@k\endcsname
4159   \expandafter\let\expandafter\BabelModifiers
4160     \csname bbl@mod@\CurrentOption\endcsname
4161   \bbl@exp{\AtBeginDocument{%
4162     \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4163   {\IfFileExists{babel-#1.tex}%
4164     {\def\bbl@tempa{%
4165       .\\There is a locale ini file for this language.\\%
4166       If it's the main language, try adding `provide=*'\\%
4167       to the babel package options}}%
4168     {\let\bbl@tempa\empty}%
4169     \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.

```

4170 \def\bbl@try@load@lang#1#2#3{%
4171   \IfFileExists{\CurrentOption.ldf}%
4172   {\bbl@load@language{\CurrentOption}}%
4173   {#1\bbl@load@language{#2}#3}}
4174 %
4175 \DeclareOption{friulian}{\bbl@try@load@lang}{\friulan}}
4176 \DeclareOption{hebrew}{%
4177   \ifcase\bbl@engine\or
4178     \bbl@error{only-pdftex-lang}{hebrew}{luatex}}%
4179 \fi
4180 \input{rlbabel.def}%
4181 \bbl@load@language{hebrew}}
4182 \DeclareOption{hungarian}{\bbl@try@load@lang}{\magyar}}
4183 \DeclareOption{lowersorbian}{\bbl@try@load@lang}{\lsorbian}}
4184 % \DeclareOption{northernkurdish}{\bbl@try@load@lang}{\kurmanji}}
4185 \DeclareOption{polutonikogreek}{%
4186   \bbl@try@load@lang}{\greek}{\languageattribute{greek}{polutoniko}}}
4187 \DeclareOption{russian}{\bbl@try@load@lang}{\russianb}}
4188 \DeclareOption{ukrainian}{\bbl@try@load@lang}{\ukraineb}}
4189 \DeclareOption{uppersorbian}{\bbl@try@load@lang}{\usorbian}}

```

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

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

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

```

4190 \ifx\GetDocumentProperties\undefined\else
4191   \let\bbl@beforeforeign\leavevmode
4192   \edef\bbl@metalang{\GetDocumentProperties{document/lang}}%
4193   \ifx\bbl@metalang\empty\else
4194     \begingroup
4195       \expandafter
4196       \bbl@bcpllookup\bbl@metalang-\empty-\empty-\empty@@
4197       \ifx\bbl@bcplrelax
4198         \ifx\bbl@opt@main\@nnil
4199           \bbl@error{no-locale-for-meta}{\bbl@metalang}{}%
4200         \fi
4201       \else
4202         \bbl@read@ini{\bbl@bcpl}\m@ne
4203         \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4204         \ifx\bbl@opt@main\@nnil
4205           \global\let\bbl@opt@main\language
4206         \fi
4207         \bbl@info{Passing \language space to babel}%
4208       \fi
4209     \endgroup
4210   \fi
4211 \fi
4212 \ifx\bbl@opt@config\@nnil
4213   \ifpackagewith{babel}{noconfigs}{}%
4214   {\InputIfFileExists{bblopts.cfg}%
4215     {\typeout{*****^J%
4216               * Local config file bblopts.cfg used^^J%
4217             *}}%
4218     }%
4219 \else
4220   \InputIfFileExists{\bbl@opt@config.cfg}%
4221   {\typeout{*****^J%
4222             * Local config file \bbl@opt@config.cfg used^^J%
4223           *}}%
4224   {\bbl@error{config-not-found}{}}}%
4225 \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).

```

4226 \def\bbl@tempf{,}
4227 \bbl@foreach\@raw@classoptionslist{%
4228   \in{=}{#1}%
4229   \ifin@ \else
4230     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \empty}%
4231   \fi}
4232 \ifx\bbl@opt@main\@nnil
4233   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4234     \let\bbl@tempb\empty
4235     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4236     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4237     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4238       \ifx\bbl@opt@main\@nnil % i.e., if not yet assigned
4239         \ifodd\bbl@iniflag % = *=
4240           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%

```

```

4241         \else % n +=
4242             \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4243         \fi
4244     \fi}%
4245 \fi
4246 \else
4247     \ifx\bbl@metalang\@undefined\else\ifx\bbl@metalang\@empty\else
4248         \bbl@afterfi\expandafter\@gobble
4249     \fi\fi % except if explicit lang metatag:
4250     {\bbl@info{Main language set with 'main='. Except if you have\\%
4251         problems, prefer the default mechanism for setting\\%
4252         the main language, i.e., as the last declared.\\%
4253         Reported}}
4254 \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).

```

4255 \ifx\bbl@opt@main\@nnil\else
4256     \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4257     \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4258 \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.

```

4259 \bbl@foreach\bbl@language@opts{%
4260     \def\bbl@tempa{#1}%
4261     \ifx\bbl@tempa\bbl@opt@main\else
4262         \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4263             \bbl@ifunset{ds@#1}%
4264             {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4265             {}%
4266         \else % + * (other = ini)
4267             \DeclareOption{#1}{%
4268                 \bbl@ldfinit
4269                 \babelprovide[@import]{#1}% %%%
4270                 \bbl@afterldf}%
4271         \fi
4272     \fi}
4273 \bbl@foreach\bbl@tempf{%
4274     \def\bbl@tempa{#1}%
4275     \ifx\bbl@tempa\bbl@opt@main\else
4276         \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4277             \bbl@ifunset{ds@#1}%
4278             {\IfFileExists{#1.ldf}%
4279              {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4280              {}}%
4281             {}%
4282         \else % + * (other = ini)
4283             \IfFileExists{babel-#1.tex}%
4284             {\DeclareOption{#1}{%
4285                 \bbl@ldfinit
4286                 \babelprovide[@import]{#1}% %%%
4287                 \bbl@afterldf}%
4288             {}}%
4289         \fi
4290     \fi}

```

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

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

```

4291 \NewHook{babel/presets}
4292 \UseHook{babel/presets}

```

```

4293 \def\AfterBabelLanguage#1{%
4294   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4295 \DeclareOption*{}
4296 \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.

```

4297 \bbl@trace{Option 'main'}
4298 \ifx\bbl@opt@main\@nnil
4299   \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4300   \let\bbl@tempc\@empty
4301   \edef\bbl@templ{\,\bbl@loaded,}
4302   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4303   \bbl@for\bbl@tempb\bbl@tempa{%
4304     \edef\bbl@tempd{\,\bbl@tempb,}%
4305     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4306     \bbl@xin{\bbl@tempd}{\bbl@templ}%
4307     \ifin\edef\bbl@tempc{\bbl@tempb}\fi
4308   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4309   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4310   \ifx\bbl@tempb\bbl@tempc\else
4311     \bbl@warning{%
4312       Last declared language option is '\bbl@tempc',\%
4313       but the last processed one was '\bbl@tempb'.\%
4314       The main language can't be set as both a global\%
4315       and a package option. Use 'main=\bbl@tempc' as\%
4316       option. Reported}
4317   \fi
4318 \else
4319   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4320     \bbl@ldfinit
4321     \let\CurrentOption\bbl@opt@main
4322     \bbl@exp{% \bbl@opt@provide = empty if *
4323       \\babelprovide
4324       [\bbl@opt@provide,@import,main]% %%%
4325       {\bbl@opt@main}}%
4326     \bbl@afterldf
4327     \DeclareOption{\bbl@opt@main}{}
4328   \else % case 0,2 (main is ldf)
4329     \ifx\bbl@loadmain\relax
4330       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4331     \else
4332       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4333     \fi
4334     \ExecuteOptions{\bbl@opt@main}
4335     \@namedef{ds@\bbl@opt@main}{}%
4336   \fi
4337   \DeclareOption*{}
4338   \ProcessOptions*
4339 \fi
4340 \bbl@exp{%
4341   \\AtBeginDocument{\\bbl@usehooks@lang{/}{begindocument}{}}}%
4342 \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.

```

4343 \ifx\bbl@main@language\@undefined
4344   \bbl@info{%
4345     You haven't specified a language as a class or package\%
4346     option. I'll load 'nil'. Reported}

```



```

4347 \bbl@load@language{nil}
4348 \fi
4349 </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`

```

4350 <*kernel>
4351 \let\bbl@onlyswitch\@empty
4352 \input babel.def
4353 \let\bbl@onlyswitch\@undefined
4354 </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.

```

4355 <*errors>
4356 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4357 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4358 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4359 \catcode`\@=11 \catcode`\^=7
4360 %
4361 \ifx\MessageBreak\@undefined
4362 \gdef\bbl@error@i#1#2{%
4363 \begingroup
4364 \newlinechar=`^^J
4365 \def\{^J(babel) }%
4366 \errhelp{#2}\errmessage{\{#1}%
4367 \endgroup}
4368 \else
4369 \gdef\bbl@error@i#1#2{%
4370 \begingroup
4371 \def\{MessageBreak}%
4372 \PackageError{babel}{#1}{#2}%
4373 \endgroup}
4374 \fi
4375 \def\bbl@errmessage#1#2#3{%
4376 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4377 \bbl@error@i{#2}{#3}}
4378 % Implicit #2#3#4:
4379 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4380 %
4381 \bbl@errmessage{not-yet-available}
4382 {Not yet available}%
4383 {Find an armchair, sit down and wait}
4384 \bbl@errmessage{bad-package-option}%
4385 {Bad option '#1=#2'. Either you have misspelled the\\%
4386 key or there is a previous setting of '#1'. Valid\\%

```

```

4387     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4388     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4389     {See the manual for further details.}
4390 \bbl@errmessage{base-on-the-fly}
4391     {For a language to be defined on the fly 'base'\\%
4392     is not enough, and the whole package must be\\%
4393     loaded. Either delete the 'base' option or\\%
4394     request the languages explicitly}%
4395     {See the manual for further details.}
4396 \bbl@errmessage{undefined-language}
4397     {You haven't defined the language '#1' yet.\\%
4398     Perhaps you misspelled it or your installation\\%
4399     is not complete}%
4400     {Your command will be ignored, type <return> to proceed}
4401 \bbl@errmessage{shorthand-is-off}
4402     {I can't declare a shorthand turned off (\string#2)}
4403     {Sorry, but you can't use shorthands which have been\\%
4404     turned off in the package options}
4405 \bbl@errmessage{not-a-shorthand}
4406     {The character '\string #1' should be made a shorthand character;\\%
4407     add the command \string\usesshorthands\string{#1\string} to
4408     the preamble.\\%
4409     I will ignore your instruction}%
4410     {You may proceed, but expect unexpected results}
4411 \bbl@errmessage{not-a-shorthand-b}
4412     {I can't switch '\string#2' on or off--not a shorthand}%
4413     {This character is not a shorthand. Maybe you made\\%
4414     a typing mistake? I will ignore your instruction.}
4415 \bbl@errmessage{unknown-attribute}
4416     {The attribute #2 is unknown for language #1.}%
4417     {Your command will be ignored, type <return> to proceed}
4418 \bbl@errmessage{missing-group}
4419     {Missing group for string \string#1}%
4420     {You must assign strings to some category, typically\\%
4421     captions or extras, but you set none}
4422 \bbl@errmessage{only-lua-xe}
4423     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4424     {Consider switching to these engines.}
4425 \bbl@errmessage{only-lua}
4426     {This macro is available only in LuaLaTeX}%
4427     {Consider switching to that engine.}
4428 \bbl@errmessage{unknown-provide-key}
4429     {Unknown key '#1' in \string\babelprovide}%
4430     {See the manual for valid keys}%
4431 \bbl@errmessage{unknown-mapfont}
4432     {Option '\bbl@KVP@mapfont' unknown for\\%
4433     mapfont. Use 'direction'}%
4434     {See the manual for details.}
4435 \bbl@errmessage{no-ini-file}
4436     {There is no ini file for the requested language\\%
4437     (#1: \language). Perhaps you misspelled it or your\\%
4438     installation is not complete}%
4439     {Fix the name or reinstall babel.}
4440 \bbl@errmessage{digits-is-reserved}
4441     {The counter name 'digits' is reserved for mapping\\%
4442     decimal digits}%
4443     {Use another name.}
4444 \bbl@errmessage{limit-two-digits}
4445     {Currently two-digit years are restricted to the\\%
4446     range 0-9999}%
4447     {There is little you can do. Sorry.}
4448 \bbl@errmessage{alphabetic-too-large}
4449     {Alphabetic numeral too large (#1)}%

```

```

4450 {Currently this is the limit.}
4451 \bbl@errmessage{no-ini-info}
4452 {I've found no info for the current locale.\\%
4453   The corresponding ini file has not been loaded\\%
4454   Perhaps it doesn't exist}%
4455 {See the manual for details.}
4456 \bbl@errmessage{unknown-ini-field}
4457 {Unknown field '#1' in \string\BCPdata.\\%
4458   Perhaps you misspelled it}%
4459 {See the manual for details.}
4460 \bbl@errmessage{unknown-locale-key}
4461 {Unknown key for locale '#2':\\%
4462   #3\\%
4463   \string#1 will be set to \string\relax}%
4464 {Perhaps you misspelled it.}%
4465 \bbl@errmessage{adjust-only-vertical}
4466 {Currently, #1 related features can be adjusted only\\%
4467   in the main vertical list}%
4468 {Maybe things change in the future, but this is what it is.}
4469 \bbl@errmessage{layout-only-vertical}
4470 {Currently, layout related features can be adjusted only\\%
4471   in vertical mode}%
4472 {Maybe things change in the future, but this is what it is.}
4473 \bbl@errmessage{bidi-only-lua}
4474 {The bidi method 'basic' is available only in\\%
4475   luatex. I'll continue with 'bidi=default', so\\%
4476   expect wrong results}%
4477 {See the manual for further details.}
4478 \bbl@errmessage{multiple-bidi}
4479 {Multiple bidi settings inside a group}%
4480 {I'll insert a new group, but expect wrong results.}
4481 \bbl@errmessage{unknown-package-option}
4482 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4483   or the language definition file \CurrentOption.ldf\\%
4484   was not found%
4485   \bbl@tempa}
4486 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4487   activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4488   headfoot=, strings=, config=, hyphenmap=, or a language name.}
4489 \bbl@errmessage{config-not-found}
4490 {Local config file '\bbl@opt@config.cfg' not found}%
4491 {Perhaps you misspelled it.}
4492 \bbl@errmessage{late-after-babel}
4493 {Too late for \string\AfterBabelLanguage}%
4494 {Languages have been loaded, so I can do nothing}
4495 \bbl@errmessage{double-hyphens-class}
4496 {Double hyphens aren't allowed in \string\babelcharclass\\%
4497   because it's potentially ambiguous}%
4498 {See the manual for further info}
4499 \bbl@errmessage{unknown-interchar}
4500 {'#1' for '\language' cannot be enabled.\\%
4501   Maybe there is a typo}%
4502 {See the manual for further details.}
4503 \bbl@errmessage{unknown-interchar-b}
4504 {'#1' for '\language' cannot be disabled.\\%
4505   Maybe there is a typo}%
4506 {See the manual for further details.}
4507 \bbl@errmessage{charproperty-only-vertical}
4508 {\string\babelcharproperty\space can be used only in\\%
4509   vertical mode (preamble or between paragraphs)}%
4510 {See the manual for further info}
4511 \bbl@errmessage{unknown-char-property}
4512 {No property named '#2'. Allowed values are\\%

```

```

4513 direction (bc), mirror (bmg), and linebreak (lb))%
4514 {See the manual for further info}
4515 \bbl@errmessage{bad-transform-option}
4516 {Bad option '#1' in a transform.\\%
4517 I'll ignore it but expect more errors}%
4518 {See the manual for further info.}
4519 \bbl@errmessage{font-conflict-transforms}
4520 {Transforms cannot be re-assigned to different\\%
4521 fonts. The conflict is in '\bbl@kv@label'.\\%
4522 Apply the same fonts or use a different label}%
4523 {See the manual for further details.}
4524 \bbl@errmessage{transform-not-available}
4525 {'#1' for '\language' cannot be enabled.\\%
4526 Maybe there is a typo or it's a font-dependent transform}%
4527 {See the manual for further details.}
4528 \bbl@errmessage{transform-not-available-b}
4529 {'#1' for '\language' cannot be disabled.\\%
4530 Maybe there is a typo or it's a font-dependent transform}%
4531 {See the manual for further details.}
4532 \bbl@errmessage{year-out-range}
4533 {Year out of range.\\%
4534 The allowed range is #1}%
4535 {See the manual for further details.}
4536 \bbl@errmessage{only-pdfTeX-lang}
4537 {The '#1' ldf style doesn't work with #2,\\%
4538 but you can use the ini locale instead.\\%
4539 Try adding 'provide=' to the option list. You may\\%
4540 also want to set 'bidi=' to some value}%
4541 {See the manual for further details.}
4542 \bbl@errmessage{hyphenmins-args}
4543 {\string\babelhyphenmins\ accepts either the optional\\%
4544 argument or the star, but not both at the same time}%
4545 {See the manual for further details.}
4546 \bbl@errmessage{no-locale-for-meta}
4547 {There isn't currently a locale for the 'lang' requested\\%
4548 in the PDF metadata ('#1'). To fix it, you can\\%
4549 set explicitly a similar language (using the same\\%
4550 script) with the key main= when loading babel. If you\\%
4551 continue, I'll fallback to the 'nil' language, with\\%
4552 tag 'und' and script 'Latn', but expect a bad font\\%
4553 rendering with other scripts. You may also need set\\%
4554 explicitly captions and date, too}%
4555 {See the manual for further details.}
4556 </errors>
4557 <*patterns>

```

8. Loading hyphenation patterns

The following code is meant to be read by $\text{\texttt{iniTeX}}$ because it should instruct $\text{\texttt{TeX}}$ 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.

```

4558 <@Make sure ProvidesFile is defined@>
4559 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4560 \xdef\bbl@format{\jobname}
4561 \def\bbl@version{<@version@>}
4562 \def\bbl@date{<@date@>}
4563 \ifx\AtBeginDocument\undefined
4564 \def\@empty{}
4565 \fi
4566 <@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.

```
4567 \def\process@line#1#2 #3 #4 {%
4568   \ifx=#1%
4569     \process@synonym{#2}%
4570   \else
4571     \process@language{#1#2}{#3}{#4}%
4572   \fi
4573   \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.

```
4574 \toks@{}
4575 \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.

```
4576 \def\process@synonym#1{%
4577   \ifnum\last@language=\m@ne
4578     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4579   \else
4580     \expandafter\chardef\curname \l@#1\endcsname\last@language
4581     \wlog{\string\l@#1=\string\language\the\last@language}%
4582     \expandafter\let\curname #1hyphenmins\expandafter\endcsname
4583     \curname\languagenname hyphenmins\endcsname
4584     \let\bbl@elt\relax
4585     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4586   \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` and `\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.

```
4587 \def\process@language#1#2#3{%
```

```

4588 \expandafter\addlanguage\csname l@#1\endcsname
4589 \expandafter\language\csname l@#1\endcsname
4590 \edef\language{#1}%
4591 \bbl@hook@everylanguage{#1}%
4592 % > luatex
4593 \bbl@get@enc#1::\@@@
4594 \begingroup
4595   \lefthyphenmin\m@ne
4596   \bbl@hook@loadpatterns{#2}%
4597   % > luatex
4598   \ifnum\lefthyphenmin=\m@ne
4599   \else
4600     \expandafter\xdef\csname #1hyphenmins\endcsname{%
4601       \the\lefthyphenmin\the\rightthyphenmin}%
4602   \fi
4603 \endgroup
4604 \def\bbl@tempa{#3}%
4605 \ifx\bbl@tempa\@empty\else
4606   \bbl@hook@loadexceptions{#3}%
4607   % > luatex
4608 \fi
4609 \let\bbl@elt\relax
4610 \edef\bbl@languages{%
4611   \bbl@languages\bbl@elt{#1}\the\language}{#2}\bbl@tempa}%
4612 \ifnum\the\language=\z@
4613   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4614     \set@hyphenmins\tw@\thr@@\relax
4615   \else
4616     \expandafter\expandafter\expandafter\set@hyphenmins
4617     \csname #1hyphenmins\endcsname
4618   \fi
4619   \the\toks@
4620   \toks@{}%
4621 \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.

```
4622 \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.

```

4623 \def\bbl@hook@everylanguage#1{}
4624 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4625 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4626 \def\bbl@hook@loadkernel#1{%
4627   \def\addlanguage{\csname newlanguage\endcsname}%
4628   \def\adddialect##1##2{%
4629     \global\chardef##1##2\relax
4630     \wlog{\string##1 = a dialect from \string\language##2}}%
4631   \def\iflanguage##1{%
4632     \expandafter\ifx\csname l@##1\endcsname\relax
4633       \nolanner{##1}%
4634     \else
4635       \ifnum\csname l@##1\endcsname=\language
4636         \expandafter\expandafter\expandafter\@firstoftwo
4637       \else
4638         \expandafter\expandafter\expandafter\@secondoftwo
4639       \fi
4640     \fi}%
4641   \def\providehyphenmins##1##2{%
4642     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax

```

```

4643 \namedef{##1hyphenmins}{##2}%
4644 \fi}%
4645 \def\set@hyphenmins##1##2{%
4646 \lefthyphenmin##1\relax
4647 \righthyphenmin##2\relax}%
4648 \def\selectlanguage{%
4649 \errhelp{Selecting a language requires a package supporting it}%
4650 \errmessage{No multilingual package has been loaded}}%
4651 \let\foreignlanguage\selectlanguage
4652 \let\otherlanguage\selectlanguage
4653 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4654 \def\bbl@usehooks##1##2{%
4655 \def\setlocale{%
4656 \errhelp{Find an armchair, sit down and wait}%
4657 \errmessage{(babel) Not yet available}}%
4658 \let\uselocale\setlocale
4659 \let\locale\setlocale
4660 \let\selectlocale\setlocale
4661 \let\localename\setlocale
4662 \let\textlocale\setlocale
4663 \let\textlanguage\setlocale
4664 \let\languagetext\setlocale}
4665 \begingroup
4666 \def\AddBabelHook#1#2{%
4667 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4668 \def\next{\toks1}%
4669 \else
4670 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4671 \fi
4672 \next}
4673 \ifx\directlua\@undefined
4674 \ifx\XeTeXinputencoding\@undefined\else
4675 \input xebabel.def
4676 \fi
4677 \else
4678 \input luababel.def
4679 \fi
4680 \openin1 = babel-\bbl@format.cfg
4681 \ifeof1
4682 \else
4683 \input babel-\bbl@format.cfg\relax
4684 \fi
4685 \closein1
4686 \endgroup
4687 \bbl@hook@loadkernel{switch.def}

```

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

```

4688 \openin1 = language.dat

```

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

```

4689 \def\language{english}%
4690 \ifeof1
4691 \message{I couldn't find the file language.dat,\space
4692 I will try the file hyphen.tex}
4693 \input hyphen.tex\relax
4694 \chardef\l@english\z@
4695 \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`.

```
4696 \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.

```
4697 \loop
4698 \endlinechar\m@ne
4699 \readl to \bbl@line
4700 \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.

```
4701 \if T\ifeoflF\fi T\relax
4702 \ifx\bbl@line\empty\else
4703 \edef\bbl@line{\bbl@line\space\space\space}%
4704 \expandafter\process@line\bbl@line\relax
4705 \fi
4706 \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.

```
4707 \begingroup
4708 \def\bbl@elt#1#2#3#4{%
4709 \global\language=#2\relax
4710 \gdef\language#1}%
4711 \def\bbl@elt##1##2##3##4{}}%
4712 \bbl@languages
4713 \endgroup
4714 \fi
4715 \closeinl
```

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

```
4716 \if/\the\toks@\else
4717 \errhelp{language.dat loads no language, only synonyms}
4718 \errmessage{Orphan language synonym}
4719 \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.

```
4720 \let\bbl@line\undefined
4721 \let\process@line\undefined
4722 \let\process@synonym\undefined
4723 \let\process@language\undefined
4724 \let\bbl@get@enc\undefined
4725 \let\bbl@hyph@enc\undefined
4726 \let\bbl@tempa\undefined
4727 \let\bbl@hook@loadkernel\undefined
4728 \let\bbl@hook@everylanguage\undefined
4729 \let\bbl@hook@loadpatterns\undefined
4730 \let\bbl@hook@loadexceptions\undefined
4731 </patterns>
```

Here the code for `initTeX` ends.

9. luatex + xetex: common stuff

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

```
4732 <<*<More package options>> ≡
4733 \chardef\bbl@bidimode\z@
```



```

4734 \DeclareOption{bidi=default}{\chardef\bb@bidimode=\@ne}
4735 \DeclareOption{bidi=basic}{\chardef\bb@bidimode=101 }
4736 \DeclareOption{bidi=basic-r}{\chardef\bb@bidimode=102 }
4737 \DeclareOption{bidi=bidi}{\chardef\bb@bidimode=201 }
4738 \DeclareOption{bidi=bidi-r}{\chardef\bb@bidimode=202 }
4739 \DeclareOption{bidi=bidi-l}{\chardef\bb@bidimode=203 }
4740 <</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. `bb@font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

```

4741 <<*Font selection>> ≡
4742 \bb@trace{Font handling with fontspec}
4743 \AddBabelHook{babel-fontspec}{afterextras}{\bb@switchfont}
4744 \AddBabelHook{babel-fontspec}{beforestart}{\bb@ckcheckstdfonts}
4745 \DisableBabelHook{babel-fontspec}
4746 \@onlypreamble\babelfont
4747 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4748   \ifx\fontspec\undefined
4749     \usepackage{fontspec}%
4750     \fi
4751     \EnableBabelHook{babel-fontspec}%
4752     \edef\bb@tempa{#1}%
4753     \def\bb@tempb{#2}% Used by \bb@bblfont
4754     \bb@bblfont}
4755 \newcommand\bb@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4756   \bb@ifunset{\bb@tempb family}%
4757     {\bb@providefam{\bb@tempb}}%
4758     {}%
4759   % For the default font, just in case:
4760   \bb@ifunset{\bb@lsys\@languagename}{\bb@provide\lsys{\@languagename}}{%
4761     \expandafter\bb@ifblank\expandafter{\bb@tempa}%
4762     {\bb@csarg\edef{\bb@tempb dflt@}{<#1>{#2}}% save \bb@rmdflt@
4763     \bb@exp{%
4764       \let\<\bb@tempb dflt@\@languagename>\<\bb@tempb dflt@>%
4765       \\\bb@font@set\<\bb@tempb dflt@\@languagename>%
4766       \<\bb@tempb default>\<\bb@tempb family>}}%
4767     {\bb@foreach\bb@tempa{% i.e., \bb@rmdflt@lang / *scrt
4768       \bb@csarg\def{\bb@tempb dflt@##1}{<#1>{#2}}}%

```

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

```

4769 \def\bb@providefam#1{%
4770   \bb@exp{%
4771     \\\newcommand\<#1default>{}% Just define it
4772     \\\bb@add@list\bb@font@fams{#1}%
4773     \\\NewHook{#1family}%
4774     \\\DeclareRobustCommand\<#1family>{%
4775       \\\not@math@alphabet\<#1family>\relax
4776       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4777       \\\fontfamily\<#1default>%
4778       \\\UseHook{#1family}%
4779       \\\selectfont}%
4780     \\\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.

```

4781 \def\bb@nostdfont#1{%
4782   \bb@ifunset{\bb@WFF@\f@family}%
4783     {\bb@csarg\gdef{WFF@\f@family}{% Flag, to avoid dupl warns
4784       \bb@infowarn{The current font is not a babel standard family:\%
4785         #1%
4786         \fontname\font\\%
4787         There is nothing intrinsically wrong with this warning, and\\%

```

```

4788     you can ignore it altogether if you do not need these\\%
4789     families. But if they are used in the document, you should be\\%
4790     aware 'babel' will not set Script and Language for them, so\\%
4791     you may consider defining a new family with \string\babelfont.\\%
4792     See the manual for further details about \string\babelfont.\\%
4793     Reported}}
4794     {}}%
4795 \gdef\bbbl@switchfont{%
4796   \bbbl@ifunset{bbbl@lsys@\languageName}{\bbbl@provide@lsys{\languageName}}{}%
4797   \bbbl@exp{% e.g., Arabic -> arabic
4798     \lowercase{\edef\\bbbl@tempa{\bbbl@cl{sname}}}}%
4799   \bbbl@foreach\bbbl@font@fams{%
4800     \bbbl@ifunset{bbbl@##1dflt@\languageName}% (1) language?
4801     {\bbbl@ifunset{bbbl@##1dflt*@\bbbl@tempa}% (2) from script?
4802       {\bbbl@ifunset{bbbl@##1dflt@}% 2=F - (3) from generic?
4803         {}% 123=F - nothing!
4804         {\bbbl@exp{% 3=T - from generic
4805           \global\let\<bbbl@##1dflt@\languageName>%
4806             \<bbbl@##1dflt@>}}}%
4807         {\bbbl@exp{% 2=T - from script
4808           \global\let\<bbbl@##1dflt@\languageName>%
4809             \<bbbl@##1dflt*@\bbbl@tempa>}}}%
4810         {}}% 1=T - language, already defined
4811   \def\bbbl@tempa{\bbbl@nostdfont{}}}%
4812   \bbbl@foreach\bbbl@font@fams{% don't gather with prev for
4813     \bbbl@ifunset{bbbl@##1dflt@\languageName}%
4814     {\bbbl@cs{famrst@##1}%
4815     \global\bbbl@csarg\let{famrst@##1}\relax}%
4816     {\bbbl@exp{% order is relevant.
4817       \\bbbl@add\\originalTeX{%
4818         \\bbbl@font@rst{\bbbl@cl{##1dflt}}%
4819         \<##1default>\<##1family>{##1}}}%
4820       \\bbbl@font@set\<bbbl@##1dflt@\languageName>% the main part!
4821       \<##1default>\<##1family>}}}%
4822   \bbbl@ifrestoring{ }\bbbl@tempa}}%

```

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

```

4823 \ifx\fbfamily\undefined\else % if latex
4824   \ifcase\bbbl@engine % if pdftex
4825     \let\bbbl@ckeckstdfonts\relax
4826   \else
4827     \def\bbbl@ckeckstdfonts{%
4828       \begingroup
4829       \global\let\bbbl@ckeckstdfonts\relax
4830       \let\bbbl@tempa\empty
4831       \bbbl@foreach\bbbl@font@fams{%
4832         \bbbl@ifunset{bbbl@##1dflt@}%
4833         {\@nameuse{##1family}}%
4834         \bbbl@csarg\gdef{WFF@\fbfamily}}}% Flag
4835         \bbbl@exp{\\bbbl@add\\bbbl@tempa{* \<##1family>= \fbfamily\\}%
4836           \space\space\fontname\font\\}%
4837         \bbbl@csarg\xdef{##1dflt@}{\fbfamily}%
4838         \expandafter\xdef\csname ##1default\endcsname{\fbfamily}}%
4839       {}}%
4840     \ifx\bbbl@tempa\empty\else
4841       \bbbl@infowarn{The following font families will use the default\\%
4842         settings for all or some languages:\\%
4843         \bbbl@tempa
4844         There is nothing intrinsically wrong with it, but\\%
4845         'babel' will no set Script and Language, which could\\%
4846         be relevant in some languages. If your document uses\\%
4847         these families, consider redefining them with \string\babelfont.\\%

```

```

4848         Reported}%
4849     \fi
4850 \endgroup}
4851 \fi
4852 \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*`).

```

4853 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4854 \bbl@xin@{<>}{#1}%
4855 \ifin@
4856 \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4857 \fi
4858 \bbl@exp{%
4859 \def\#2\#1{% e.g., \rmdefault{\bbl@rmdflt@lang}
4860 \bbl@ifsamestring{#2}{\f@family}%
4861 {\#3%
4862 \bbl@ifsamestring{\f@series}{\bfdefault}{\bfbseries}{}%
4863 \let\bbl@tempa\relax}%
4864 }}}

```

Loaded locally, which does its job, but very must be global. The problem is how. This actually defines a font predeclared with `\babelfont`, making sure Script and Language names are defined. If they are not, the corresponding data in the ini file is used. The font is actually set temporarily to get the family name (`\f@family`). There is also a hack because by default some replacements related to the bold series are sometimes assigned to the wrong font (see issue #92).

```

4865 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4866 \let\bbl@tempe\bbl@mapselect
4867 \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
4868 \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}{}}%
4869 \let\bbl@mapselect\relax
4870 \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
4871 \let#4\@empty % Make sure \renewfontfamily is valid
4872 \bbl@set@renderer
4873 \bbl@exp{%
4874 \let\bbl@temp@pfam<\bbl@stripslash#4\space>% e.g., '\rmfamily'
4875 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4876 {\bbl@newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4877 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4878 {\bbl@newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4879 \bbl@renewfontfamily\#4%
4880 [\bbl@cl{sys},% xetex removes unknown features :-(
4881 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4882 #2]}{#3}% i.e., \bbl@exp{..}{#3}
4883 \bbl@unset@renderer
4884 \begingroup
4885 #4%
4886 \xdef#1{\f@family}% e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4887 \endgroup
4888 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4889 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4890 \ifin@
4891 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4892 \fi
4893 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4894 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%

```

```

4895 \ifin@
4896 \global\bbbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4897 \fi
4898 \let#4\bbbl@temp@fam
4899 \bbbl@exp{\let\<\bbbl@stripslash#4\space>}\bbbl@temp@pfam
4900 \let\bbbl@mapselect\bbbl@tempe}%

```

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

```

4901 \def\bbbl@font@rst#1#2#3#4{%
4902 \bbbl@csarg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}

```

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

```

4903 \def\bbbl@font@fams{rm,sf,tt}
4904 <</Font selection>>

```

10. Hooks for XeTeX and LuaTeX

10.1. XeTeX

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

Now, the code.

```

4905 <*<xtex>
4906 \def\BabelStringsDefault{unicode}
4907 \let\xebbl@stop\relax
4908 \AddBabelHook{xetex}{encodedcommands}{%
4909 \def\bbbl@tempa{#1}%
4910 \ifx\bbbl@tempa\@empty
4911 \XeTeXinputencoding"bytes"%
4912 \else
4913 \XeTeXinputencoding"#1"%
4914 \fi
4915 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4916 \AddBabelHook{xetex}{stopcommands}{%
4917 \xebbl@stop
4918 \let\xebbl@stop\relax}
4919 \def\bbbl@input@classes{% Used in CJK intraspaces
4920 \input{load-unicode-xetex-classes.tex}%
4921 \let\bbbl@input@classes\relax}
4922 \def\bbbl@intraspace#1 #2 #3\@@{%
4923 \bbbl@csarg\gdef{xeisp@\languagename}%
4924 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4925 \def\bbbl@intrapenalty#1\@@{%
4926 \bbbl@csarg\gdef{xeipn@\languagename}%
4927 {\XeTeXlinebreakpenalty #1\relax}}
4928 \def\bbbl@provide@intraspace{%
4929 \bbbl@xin@{/s}{\bbbl@cl{lnbrk}}}%
4930 \ifin@else\bbbl@xin@{/c}{\bbbl@cl{lnbrk}}\fi
4931 \ifin@
4932 \bbbl@ifunset{bbbl@intsp@\languagename}{}%
4933 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4934 \ifx\bbbl@KVP@intraspace\@nnil
4935 \bbbl@exp{%
4936 \\\bbbl@intraspace\bbbl@cl{intsp}\\\@@}%
4937 \fi
4938 \ifx\bbbl@KVP@intrapenalty\@nnil
4939 \bbbl@intrapenalty0\@@
4940 \fi
4941 \fi
4942 \ifx\bbbl@KVP@intraspace\@nnil\else % We may override the ini

```

```

4943 \expandafter\bbbl@intraspace\bbbl@KVP@intraspace\@@
4944 \fi
4945 \ifx\bbbl@KVP@intrapenalty\@nnil\else
4946 \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
4947 \fi
4948 \bbbl@exp{%
4949 \\\bbbl@add\<extras\language>{%
4950 \XeTeXlinebreaklocale "\bbbl@cl{tbc}"%
4951 \<bbbl@xeisp@\language>%
4952 \<bbbl@xeipn@\language>%
4953 \\\bbbl@tglobal\<extras\language>%
4954 \\\bbbl@add\<noextras\language>{%
4955 \XeTeXlinebreaklocale ""}%
4956 \\\bbbl@tglobal\<noextras\language>%
4957 \ifx\bbbl@ispace\@undefined
4958 \gdef\bbbl@ispace{\bbbl@cl{xeisp}}%
4959 \ifx\AtBeginDocument\@notprerr
4960 \expandafter\@secondoftwo % to execute right now
4961 \fi
4962 \AtBeginDocument{\bbbl@patchfont{\bbbl@ispace}}%
4963 \fi}%
4964 \fi}
4965 \ifx\DisableBabelHook\@undefined\endinput\fi
4966 \let\bbbl@set@renderer\relax
4967 \let\bbbl@unset@renderer\relax
4968 <@Font selection@>
4969 \def\bbbl@provide@extra#1{}

```

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

```

4970 \def\bbbl@xenohyph@d{%
4971 \bbbl@ifset{\bbbl@prehc@\language}%
4972 {\ifnum\hyphenchar\font=\defaultthyphenchar
4973 \iffontchar\font\bbbl@cl{prehc}\relax
4974 \hyphenchar\font\bbbl@cl{prehc}\relax
4975 \else\iffontchar\font"200B
4976 \hyphenchar\font"200B
4977 \else
4978 \bbbl@warning
4979 {Neither 0 nor ZERO WIDTH SPACE are available\\%
4980 in the current font, and therefore the hyphen\\%
4981 will be printed. Try changing the fontspec's\\%
4982 'HyphenChar' to another value, but be aware\\%
4983 this setting is not safe (see the manual).\\%
4984 Reported}%
4985 \hyphenchar\font\defaultthyphenchar
4986 \fi\fi
4987 \fi}%
4988 {\hyphenchar\font\defaultthyphenchar}}

```

10.2. Support for interchar

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

```

4989 \ifnum\xe@alloc@intercharclass<\thr@@
4990 \xe@alloc@intercharclass\thr@@
4991 \fi
4992 \chardef\bbbl@xe@class@default@=\z@
4993 \chardef\bbbl@xe@class@cjkideogram@=\@ne
4994 \chardef\bbbl@xe@class@cjkleftpunctuation@=\tw@
4995 \chardef\bbbl@xe@class@cjkrightpunctuation@=\thr@@
4996 \chardef\bbbl@xe@class@boundary@=4095
4997 \chardef\bbbl@xe@class@ignore@=4096

```

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

```

4998 \AddBabelHook{babel-interchar}{beforeextras}{%
4999   \@nameuse{bbl@xechars@\language}\relax}%
5000 \DisableBabelHook{babel-interchar}
5001 \protected\def\bbl@charclass#1{%
5002   \ifnum\count@<\z@
5003     \count@=-\count@
5004     \loop
5005       \bbl@exp{%
5006         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5007         \XeTeXcharclass\count@ \bbl@tempc
5008         \ifnum\count@<`#1\relax
5009           \advance\count@\@ne
5010         \repeat
5011   \else
5012     \babel@savevariable{\XeTeXcharclass`#1}%
5013     \XeTeXcharclass`#1 \bbl@tempc
5014   \fi
5015   \count@`#1\relax}

```

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

```

5016 \newcommand\bbl@ifinterchar[1]{%
5017   \let\bbl@tempa\@gobble % Assume to ignore
5018   \edef\bbl@tempb{\zap@space#1 \@empty}%
5019   \ifx\bbl@KVP@interchar\@nnil\else
5020     \bbl@replace\bbl@KVP@interchar{ }{,}%
5021     \bbl@foreach\bbl@tempb{%
5022       \bbl@xin{,##1,}{, \bbl@KVP@interchar,}%
5023       \ifin@
5024         \let\bbl@tempa\@firstofone
5025       \fi}%
5026   \fi
5027   \bbl@tempa}
5028 \newcommand\IfBabelIntercharT[2]{%
5029   \bbl@carg\bbl@add{\bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5030 \newcommand\babelcharclass[3]{%
5031   \EnableBabelHook{babel-interchar}%
5032   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5033   \def\bbl@tempb#1{%
5034     \ifx##1\@empty\else
5035       \ifx##1-%
5036         \bbl@upto
5037       \else
5038         \bbl@charclass{%
5039           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5040         \fi
5041       \expandafter\bbl@tempb
5042     \fi}%
5043   \bbl@ifunset{bbl@xechars@#1}%
5044   {\toks@{%
5045     \babel@savevariable\XeTeXinterchartokenstate
5046     \XeTeXinterchartokenstate\@ne
5047   }}%
5048   {\toks@\expandafter\expandafter\expandafter{%
5049     \csname bbl@xechars@#1\endcsname}}}%

```

```

5050 \bbl@csarg\edef{xechars@#1}{%
5051 \the\toks@
5052 \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5053 \bbl@tempb#3\@empty}}
5054 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
5055 \protected\def\bbl@upto{%
5056 \ifnum\count@>\z@
5057 \advance\count@\@ne
5058 \count@-\count@
5059 \else\ifnum\count@=\z@
5060 \bbl@charclass{-}%
5061 \else
5062 \bbl@error{double-hyphens-class}{\}\}%
5063 \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>.

5064 \def\bbl@ignoreinterchar{%
5065 \ifnum\language=\l@nohyphenation
5066 \expandafter\@gobble
5067 \else
5068 \expandafter\@firstofone
5069 \fi}
5070 \newcommand\babelinterchar[5][\]{%
5071 \let\bbl@kv@label\@empty
5072 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5073 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5074 {\bbl@ignoreinterchar{#5}}%
5075 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5076 \bbl@expf{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5077 \bbl@expf{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5078 \XeTeXinterchartoks
5079 \@nameuse{bbl@xeclass@\bbl@tempa @#2}\@firstofone
5080 \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{\}\}%
5081 \@nameuse{bbl@xeclass@\bbl@tempb @#2}\@firstofone
5082 \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{\}\}%
5083 = \expandafter{
5084 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5085 \csname\zap@space bbl@xeinter@\bbl@kv@label
5086 @#3@#4@#2 \@empty\endcsname}}}}
5087 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5088 \bbl@ifunset{bbl@ic@#1@languagename}%
5089 {\bbl@error{unknown-interchar}{#1}{\}\}%
5090 {\bbl@csarg\let{ic@#1@languagename}\@firstofone}}
5091 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5092 \bbl@ifunset{bbl@ic@#1@languagename}%
5093 {\bbl@error{unknown-interchar-b}{#1}{\}\}%
5094 {\bbl@csarg\let{ic@#1@languagename}\@gobble}}
5095 </xetex>

```

10.3. Layout

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

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

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

```

5096 < *xetex | texxet >
5097 \providecommand\bbl@provide@intraspace{}
5098 \bbl@trace{Redefinitions for bidi layout}

```

Finish here if there in no layout.

```
5099 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5100 \IfBabelLayout{nopars}
5101 {}
5102 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5103 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5104 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5105 \ifnum\bbl@bidimode>\z@
5106 \IfBabelLayout{pars}
5107 {\def\@hangfrom#1{%
5108   \setbox\@tempboxa\hbox{#{#1}}%
5109   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5110   \noindent\box\@tempboxa}
5111 \def\raggedright{%
5112   \let\\\@centercr
5113   \bbl@startskip\z@skip
5114   \@rightskip\@flushglue
5115   \bbl@endskip\@rightskip
5116   \parindent\z@
5117   \parfillskip\bbl@startskip}
5118 \def\raggedleft{%
5119   \let\\\@centercr
5120   \bbl@startskip\@flushglue
5121   \bbl@endskip\z@skip
5122   \parindent\z@
5123   \parfillskip\bbl@endskip}}
5124 {}
5125 \fi
5126 \IfBabelLayout{lists}
5127 {\bbl@sreplace\list
5128   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5129   \def\bbl@listleftmargin{%
5130     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5131   \ifcase\bbl@engine
5132     \def\labelenumii{}\theenumii{}\% pdfTeX doesn't reverse ()
5133     \def\p@enumiii{\p@enumii}\theenumii{}\%
5134   \fi
5135   \bbl@sreplace\@verbatim
5136     {\leftskip\@totalleftmargin}%
5137     {\bbl@startskip\textwidth
5138       \advance\bbl@startskip-\linewidth}%
5139   \bbl@sreplace\@verbatim
5140     {\rightskip\z@skip}%
5141     {\bbl@endskip\z@skip}}%
5142 {}
5143 \IfBabelLayout{contents}
5144 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5145   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5146 {}
5147 \IfBabelLayout{columns}
5148 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5149   \def\bbl@outputbox#1{%
5150     \hb@xt@\textwidth{%
5151       \hskip\columnwidth
5152       \hfil
5153       {\normalcolor\vrule \@width\columnseprule}%
5154       \hfil
5155       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5156       \hskip-\textwidth
5157       \hb@xt@\columnwidth{\box\@outputbox \hss}%
5158       \hskip\columnsep
5159       \hskip\columnwidth}}}%
5160 {}
```


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.

```

5161 \IfBabelLayout{counters*}%
5162   {\bbl@add\bbl@opt@layout{.counters.}%
5163     \AddToHook{shipout/before}{%
5164       \let\bbl@tempa\babelsublr
5165       \let\babelsublr\@firstofone
5166       \let\bbl@save@thepage\thepage
5167       \protected@edef\thepage{\thepage}%
5168       \let\babelsublr\bbl@tempa}%
5169     \AddToHook{shipout/after}{%
5170       \let\thepage\bbl@save@thepage}}{}
5171 \IfBabelLayout{counters}%
5172   {\let\bbl@latinarabic=\@arabic
5173     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5174     \let\bbl@asciroman=\@roman
5175     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5176     \let\bbl@asciiRoman=\@Roman
5177     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5178 \fi % end if layout
5179 </xetex | texpet>

```

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).

```

5180 < *texpet>
5181 \def\bbl@provide@extra#1{%
5182   % == auto-select encoding ==
5183   \ifx\bbl@encoding@select@off\@empty\else
5184     \bbl@ifunset{\bbl@encoding@#1}%
5185     {\def\@elt##1{,##1,}%
5186       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5187       \count@z@
5188       \bbl@foreach\bbl@tempe{%
5189         \def\bbl@tempd{##1}% Save last declared
5190         \advance\count@\@ne}%
5191       \ifnum\count@>\@ne % (1)
5192         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5193         \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5194         \bbl@replace\bbl@tempa{ },}%
5195         \global\bbl@csarg\let{encoding@#1}\@empty
5196         \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5197         \ifin\else % if main encoding included in ini, do nothing
5198           \let\bbl@tempb\relax
5199           \bbl@foreach\bbl@tempa{%
5200             \ifx\bbl@tempb\relax
5201               \bbl@xin@{,##1,}{,\bbl@tempe,}%
5202               \ifin\def\bbl@tempb{##1}\fi
5203             \fi}%
5204           \ifx\bbl@tempb\relax\else
5205             \bbl@exp{%
5206               \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>%
5207               \gdef\<\bbl@encoding@#1>{%
5208                 \\babel@save\\f@encoding
5209                 \\bbl@add\\originalTeX{\\selectfont}%
5210                 \\fontencoding{\bbl@tempb}%
5211                 \\selectfont}}%
5212             \fi
5213           \fi
5214         \fi}%
5215       }%

```

```

5216 \fi}
5217 </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`).

```

5218 <*luatex>
5219 \directlua{ Babel = Babel or {} } % DL2
5220 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5221 \bbl@trace{Read language.dat}
5222 \ifx\bbl@readstream\undefined
5223 \csname newread\endcsname\bbl@readstream
5224 \fi
5225 \begingroup
5226 \toks@{}
5227 \count@ \z@ % 0=start, 1=0th, 2=normal
5228 \def\bbl@process@line#1#2 #3 #4 {%
5229 \ifx=#1%
5230 \bbl@process@synonym{#2}%
5231 \else
5232 \bbl@process@language{#1#2}{#3}{#4}%
5233 \fi
5234 \ignorespaces}
5235 \def\bbl@manylang{%
5236 \ifnum\bbl@last>\@ne
5237 \bbl@info{Non-standard hyphenation setup}%
5238 \fi
5239 \let\bbl@manylang\relax}
5240 \def\bbl@process@language#1#2#3{%
5241 \ifcase\count@
5242 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5243 \or

```

```

5244     \count@ \tw@
5245 \fi
5246 \ifnum \count@ = \tw@
5247     \expandafter \addlanguage \csname l@#1 \endcsname
5248     \language \allocationnumber
5249     \chardef \bbl@last \allocationnumber
5250     \bbl@many lang
5251     \let \bbl@elt \relax
5252     \xdef \bbl@languages {%
5253         \bbl@languages \bbl@elt{#1}{\the \language}{#2}{#3}}%
5254 \fi
5255 \the \toks@
5256 \toks@ {}
5257 \def \bbl@process@synonym@aux#1#2{%
5258     \global \expandafter \chardef \csname l@#1 \endcsname #2 \relax
5259     \let \bbl@elt \relax
5260     \xdef \bbl@languages {%
5261         \bbl@languages \bbl@elt{#1}{#2}{}}}%
5262 \def \bbl@process@synonym#1{%
5263     \ifcase \count@
5264         \toks@ \expandafter {\the \toks@ \relax \bbl@process@synonym{#1}}%
5265     \or
5266         \ifundefined {zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5267     \else
5268         \bbl@process@synonym@aux{#1}{\the \bbl@last}%
5269     \fi
5270 \ifx \bbl@languages \@undefined % Just a (sensible?) guess
5271     \chardef \l@english \z@
5272     \chardef \l@USenglish \z@
5273     \chardef \bbl@last \z@
5274     \global \@namedef {bbl@hyphendata@0}{{\hyphen.tex}}
5275     \gdef \bbl@languages {%
5276         \bbl@elt{english}{0}{\hyphen.tex}}%
5277     \bbl@elt{USenglish}{0}{}
5278 \else
5279     \global \let \bbl@languages @format \bbl@languages
5280     \def \bbl@elt#1#2#3#4{% Remove all except language 0
5281         \ifnum #2 > \z@ \else
5282             \noexpand \bbl@elt{#1}{#2}{#3}{#4}%
5283         \fi}%
5284     \xdef \bbl@languages {\bbl@languages}%
5285 \fi
5286 \def \bbl@elt#1#2#3#4{\@namedef {zth@#1}} % Define flags
5287 \bbl@languages
5288 \openin \bbl@readstream = language.dat
5289 \ifeof \bbl@readstream
5290     \bbl@warning {I couldn't find language.dat. No additional \\\%
5291         patterns loaded. Reported}%
5292 \else
5293     \loop
5294         \endlinechar \m@ne
5295         \read \bbl@readstream to \bbl@line
5296         \endlinechar ``^M
5297         \if T \ifeof \bbl@readstream F \fi T \relax
5298         \ifx \bbl@line \@empty \else
5299             \edef \bbl@line {\bbl@line \space \space \space}%
5300             \expandafter \bbl@process@line \bbl@line \relax
5301         \fi
5302     \repeat
5303 \fi
5304 \closein \bbl@readstream
5305 \endgroup
5306 \bbl@trace {Macros for reading patterns files}

```

```

5307 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
5308 \ifx\babelcatcodetablenum\undefined
5309 \ifx\newcatcodetable\undefined
5310 \def\babelcatcodetablenum{5211}
5311 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5312 \else
5313 \newcatcodetable\babelcatcodetablenum
5314 \newcatcodetable\bbl@pattcodes
5315 \fi
5316 \else
5317 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5318 \fi
5319 \def\bbl@luapatterns#1#2{%
5320 \bbl@get@enc#1::\@@@
5321 \setbox\z@\hbox\bgroup
5322 \begingroup
5323 \savecatcodetable\babelcatcodetablenum\relax
5324 \initcatcodetable\bbl@pattcodes\relax
5325 \catcodetable\bbl@pattcodes\relax
5326 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5327 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5328 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5329 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5330 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5331 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5332 \input #1\relax
5333 \catcodetable\babelcatcodetablenum\relax
5334 \endgroup
5335 \def\bbl@tempa{#2}%
5336 \ifx\bbl@tempa\empty\else
5337 \input #2\relax
5338 \fi
5339 \egroup}%
5340 \def\bbl@patterns@lua#1{%
5341 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5342 \csname l@#1\endcsname
5343 \edef\bbl@tempa{#1}%
5344 \else
5345 \csname l@#1:f@encoding\endcsname
5346 \edef\bbl@tempa{#1:f@encoding}%
5347 \fi\relax
5348 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5349 \@ifundefined{bbl@hyphendata@the\language}%
5350 {\def\bbl@elt##1##2##3##4{%
5351 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5352 \def\bbl@tempb{##3}%
5353 \ifx\bbl@tempb\empty\else % if not a synonymous
5354 \def\bbl@tempc{{##3}{##4}}%
5355 \fi
5356 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5357 \fi}%
5358 \bbl@languages
5359 \@ifundefined{bbl@hyphendata@the\language}%
5360 {\bbl@info{No hyphenation patterns were set for\%
5361 language '\bbl@tempa'. Reported}}%
5362 {\expandafter\expandafter\expandafter\bbl@luapatterns
5363 \csname bbl@hyphendata@the\language\endcsname}}}%
5364 \endinput\fi

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

5365 \ifx\DisableBabelHook\@undefined
5366 \AddBabelHook{luatex}{everylanguage}{%
5367 \def\process@language##1##2##3{%

```

```

5368     \def\process@line####1####2 ####3 ####4 {}}}
5369 \AddBabelHook{luatex}{loadpatterns}{%
5370     \input #1\relax
5371     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5372         {{#1}}}}
5373 \AddBabelHook{luatex}{loadexceptions}{%
5374     \input #1\relax
5375     \def\bbl@tempb##1##2{{##1}{##2}}%
5376     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5377         {\expandafter\expandafter\expandafter\bbl@tempb
5378         \csname bbl@hyphendata@the\language\endcsname}}
5379 \endinput\fi

```

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

```

5380 \begingroup
5381 \catcode`\%=12
5382 \catcode`\'=12
5383 \catcode`\-=12
5384 \catcode`\:=12
5385 \directlua{
5386     Babel.locale_props = Babel.locale_props or {}
5387     function Babel.lua_error(e, a)
5388         tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5389             e .. '}' .. (a or '') .. '}{}{}')
5390     end
5391
5392     function Babel.bytes(line)
5393         return line:gsub(".",
5394             function (chr) return unicode.utf8.char(string.byte(chr)) end)
5395     end
5396
5397     function Babel.priority_in_callback(name,description)
5398         for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5399             if v == description then return i end
5400         end
5401         return false
5402     end
5403
5404     function Babel.begin_process_input()
5405         if luatexbase and luatexbase.add_to_callback then
5406             luatexbase.add_to_callback('process_input_buffer',
5407                 Babel.bytes,'Babel.bytes')
5408         else
5409             Babel.callback = callback.find('process_input_buffer')
5410             callback.register('process_input_buffer',Babel.bytes)
5411         end
5412     end
5413
5414     function Babel.end_process_input ()
5415         if luatexbase and luatexbase.remove_from_callback then
5416             luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5417         else
5418             callback.register('process_input_buffer',Babel.callback)
5419         end
5420     end
5421
5422     function Babel.str_to_nodes(fn, matches, base)
5423         local n, head, last
5424         if fn == nil then return nil end
5425         for s in string.utfvalues(fn(matches)) do
5426             if base.id == 7 then
5427                 base = base.replace

```

```

5428     n = node.copy(base)
5429     n.char = s
5430     if not head then
5431         head = n
5432     else
5433         last.next = n
5434     end
5435     last = n
5436 end
5437 return head
5438 end
5439
5440 Babel.linebreaking = Babel.linebreaking or {}
5441 Babel.linebreaking.before = {}
5442 Babel.linebreaking.after = {}
5443 Babel.locale = {}
5444 function Babel.linebreaking.add_before(func, pos)
5445     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5446     if pos == nil then
5447         table.insert(Babel.linebreaking.before, func)
5448     else
5449         table.insert(Babel.linebreaking.before, pos, func)
5450     end
5451 end
5452 function Babel.linebreaking.add_after(func)
5453     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5454     table.insert(Babel.linebreaking.after, func)
5455 end
5456
5457 function Babel.addpatterns(pp, lg)
5458     local lg = lang.new(lg)
5459     local pats = lang.patterns(lg) or ''
5460     lang.clear_patterns(lg)
5461     for p in pp:gmatch('[^%s]+') do
5462         ss = ''
5463         for i in string.utfcharacters(p:gsub('%d', '')) do
5464             ss = ss .. '%d?' .. i
5465         end
5466         ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5467         ss = ss:gsub('%.%d%?$', '%%.')
5468         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5469         if n == 0 then
5470             tex.sprint(
5471                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5472                 .. p .. [[{ }]])
5473             pats = pats .. ' ' .. p
5474         else
5475             tex.sprint(
5476                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5477                 .. p .. [[{ }]])
5478         end
5479     end
5480     lang.patterns(lg, pats)
5481 end
5482
5483 Babel.characters = Babel.characters or {}
5484 Babel.ranges = Babel.ranges or {}
5485 function Babel.hlist_has_bidi(head)
5486     local has_bidi = false
5487     local ranges = Babel.ranges
5488     for item in node.traverse(head) do
5489         if item.id == node.id'glyph' then
5490             local itemchar = item.char

```

```

5491     local chardata = Babel.characters[itemchar]
5492     local dir = chardata and chardata.d or nil
5493     if not dir then
5494         for nn, et in ipairs(ranges) do
5495             if itemchar < et[1] then
5496                 break
5497             elseif itemchar <= et[2] then
5498                 dir = et[3]
5499                 break
5500             end
5501         end
5502     end
5503     if dir and (dir == 'al' or dir == 'r') then
5504         has_bidi = true
5505     end
5506 end
5507 end
5508 return has_bidi
5509 end
5510 function Babel.set_chrnges_b (script, chrng)
5511     if chrng == '' then return end
5512     texio.write('Replacing ' .. script .. ' script ranges')
5513     Babel.script_blocks[script] = {}
5514     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5515         table.insert(
5516             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5517     end
5518 end
5519
5520 function Babel.discard_sublr(str)
5521     if str:find( [[\string\indexentry]] ) and
5522        str:find( [[\string\babelsublr]] ) then
5523         str = str:gsub( [[\string\babelsublr%*(%b{})]],
5524                        function(m) return m:sub(2,-2) end )
5525     end
5526     return str
5527 end
5528 }
5529 \endgroup
5530 \ifx\newattribute\@undefined\else % Test for plain
5531     \newattribute\bbl@attr@locale % DL4
5532     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5533     \AddBabelHook{luatex}{beforeextras}{%
5534         \setattribute\bbl@attr@locale\localeid}
5535 \fi
5536 %
5537 \def\BabelStringsDefault{unicode}
5538 \let\luabbbl@stop\relax
5539 \AddBabelHook{luatex}{encodedcommands}{%
5540     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5541     \ifx\bbl@tempa\bbl@tempb\else
5542         \directlua{Babel.begin_process_input()}%
5543         \def\luabbbl@stop{%
5544             \directlua{Babel.end_process_input()}}%
5545     \fi}%
5546 \AddBabelHook{luatex}{stopcommands}{%
5547     \luabbbl@stop
5548     \let\luabbbl@stop\relax}
5549 %
5550 \AddBabelHook{luatex}{patterns}{%
5551     \@ifundefined{bbl@hyphendata@the\language}%
5552     {\def\bbl@elt##1##2##3##4{%
5553         \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...

```

```

5554 \def\bbl@tempb{##3}%
5555 \ifx\bbl@tempb\@empty\else % if not a synonymous
5556 \def\bbl@tempc{##3}{##4}%
5557 \fi
5558 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5559 \fi}%
5560 \bbl@languages
5561 \@ifundefined{bbl@hyphendata@the\language}%
5562 {\bbl@info{No hyphenation patterns were set for\%
5563 language '#2'. Reported}}%
5564 {\expandafter\expandafter\expandafter\bbl@luapatterns
5565 \csname bbl@hyphendata@the\language\endcsname}}}%
5566 \@ifundefined{bbl@patterns@}{}%
5567 \begingroup
5568 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5569 \ifin@else
5570 \ifx\bbl@patterns@\@empty\else
5571 \directlua{ Babel.addpatterns(
5572 [[\bbl@patterns@]], \number\language) }%
5573 \fi
5574 \@ifundefined{bbl@patterns@#1}%
5575 \@empty
5576 {\directlua{ Babel.addpatterns(
5577 [[\space\csname bbl@patterns@#1\endcsname]],
5578 \number\language) }}%
5579 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5580 \fi
5581 \endgroup}%
5582 \bbl@exp{%
5583 \bbl@ifunset{bbl@prehc@language\name}{}%
5584 {\bbl@ifblank{\bbl@cs{prehc@language\name}}}%
5585 {\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.

```

5586 \@onlypreamble\babelpatterns
5587 \AtEndOfPackage{%
5588 \newcommand\babelpatterns[2][\@empty]{%
5589 \ifx\bbl@patterns@\relax
5590 \let\bbl@patterns@\@empty
5591 \fi
5592 \ifx\bbl@pttnlist@\@empty\else
5593 \bbl@warning{%
5594 You must not intermingle \string\selectlanguage\space and\%
5595 \string\babelpatterns\space or some patterns will not\%
5596 be taken into account. Reported}%
5597 \fi
5598 \ifx@\empty#1%
5599 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5600 \else
5601 \edef\bbl@tempb{\zap@space#1 \@empty}%
5602 \bbl@for\bbl@tempa\bbl@tempb{%
5603 \bbl@fixname\bbl@tempa
5604 \bbl@iflanguage\bbl@tempa{%
5605 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5606 \@ifundefined{bbl@patterns@\bbl@tempa}%
5607 \@empty
5608 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5609 #2}}}%
5610 \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.

```
5611 \def\bbl@intraspace#1 #2 #3\@@{%
5612   \directlua{
5613     Babel.intraspaces = Babel.intraspaces or {}
5614     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5615       {b = #1, p = #2, m = #3}
5616     Babel.locale_props[\the\localeid].intraspace = %
5617       {b = #1, p = #2, m = #3}
5618   }}
5619 \def\bbl@intrapenalty#1\@@{%
5620   \directlua{
5621     Babel.intrapenalties = Babel.intrapenalties or {}
5622     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5623     Babel.locale_props[\the\localeid].intrapenalty = #1
5624   }}
5625 \begingroup
5626 \catcode`\%=12
5627 \catcode`\&=14
5628 \catcode`\'=12
5629 \catcode`\-=12
5630 \gdef\bbl@seaintraspace{&
5631   \let\bbl@seaintraspace\relax
5632   \directlua{
5633     Babel.sea_enabled = true
5634     Babel.sea_ranges = Babel.sea_ranges or {}
5635     function Babel.set_chranges (script, chrng)
5636       local c = 0
5637       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5638         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5639         c = c + 1
5640       end
5641     end
5642     function Babel.sea_disc_to_space (head)
5643       local sea_ranges = Babel.sea_ranges
5644       local last_char = nil
5645       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5646       for item in node.traverse(head) do
5647         local i = item.id
5648         if i == node.id'glyph' then
5649           last_char = item
5650         elseif i == 7 and item.subtype == 3 and last_char
5651           and last_char.char > 0x0C99 then
5652           quad = font.getfont(last_char.font).size
5653           for lg, rg in pairs(sea_ranges) do
5654             if last_char.char > rg[1] and last_char.char < rg[2] then
5655               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5656               local intraspace = Babel.intraspaces[lg]
5657               local intrapenalty = Babel.intrapenalties[lg]
5658               local n
5659               if intrapenalty ~= 0 then
5660                 n = node.new(14, 0)      &% penalty
5661                 n.penalty = intrapenalty
5662                 node.insert_before(head, item, n)
5663               end
5664               n = node.new(12, 13)      &% (glue, spaceskip)
5665               node.setglue(n, intraspace.b * quad,
5666                 intraspace.p * quad,
5667                 intraspace.m * quad)
```

```

5668         node.insert_before(head, item, n)
5669         node.remove(head, item)
5670     end
5671 end
5672 end
5673 end
5674 end
5675 }&
5676 \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.

```

5677 \catcode`\%=14
5678 \gdef\bbl@cjkintraspace{%
5679   \let\bbl@cjkintraspace\relax
5680   \directlua{
5681     require('babel-data-cjk.lua')
5682     Babel.cjk_enabled = true
5683     function Babel.cjk_linebreak(head)
5684       local GLYPH = node.id'glyph'
5685       local last_char = nil
5686       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5687       local last_class = nil
5688       local last_lang = nil
5689       for item in node.traverse(head) do
5690         if item.id == GLYPH then
5691           local lang = item.lang
5692           local LOCALE = node.get_attribute(item,
5693             Babel.attr_locale)
5694           local props = Babel.locale_props[LOCALE] or {}
5695           local class = Babel.cjk_class[item.char].c
5696           if props.cjk_quotes and props.cjk_quotes[item.char] then
5697             class = props.cjk_quotes[item.char]
5698           end
5699           if class == 'cp' then class = 'cl' % )] as CL
5700           elseif class == 'id' then class = 'I'
5701           elseif class == 'cj' then class = 'I' % loose
5702           end
5703           local br = 0
5704           if class and last_class and Babel.cjk_breaks[last_class][class] then
5705             br = Babel.cjk_breaks[last_class][class]
5706           end
5707           if br == 1 and props.linebreak == 'c' and
5708             lang ~= \the\l@nohyphenation\space and
5709             last_lang ~= \the\l@nohyphenation then
5710             local intrapenalty = props.intrapenalty
5711             if intrapenalty ~= 0 then
5712               local n = node.new(14, 0)      % penalty
5713               n.penalty = intrapenalty
5714               node.insert_before(head, item, n)
5715             end
5716             local intraspace = props.intraspace
5717             local n = node.new(12, 13)      % (glue, spaceskip)
5718             node.setglue(n, intraspace.b * quad,
5719               intraspace.p * quad,
5720               intraspace.m * quad)
5721             node.insert_before(head, item, n)

```

```

5722         end
5723         if font.getfont(item.font) then
5724             quad = font.getfont(item.font).size
5725         end
5726         last_class = class
5727         last_lang = lang
5728         else % if penalty, glue or anything else
5729             last_class = nil
5730         end
5731     end
5732     lang.hyphenate(head)
5733 end
5734 }%
5735 \bbl@luahyphenate}
5736 \gdef\bbl@luahyphenate{%
5737 \let\bbl@luahyphenate\relax
5738 \directlua{
5739     luatexbase.add_to_callback('hyphenate',
5740     function (head, tail)
5741         if Babel.linebreaking.before then
5742             for k, func in ipairs(Babel.linebreaking.before) do
5743                 func(head)
5744             end
5745         end
5746         lang.hyphenate(head)
5747         if Babel.cjk_enabled then
5748             Babel.cjk_linebreak(head)
5749         end
5750         if Babel.linebreaking.after then
5751             for k, func in ipairs(Babel.linebreaking.after) do
5752                 func(head)
5753             end
5754         end
5755         if Babel.set_hboxed then
5756             Babel.set_hboxed(head)
5757         end
5758         if Babel.sea_enabled then
5759             Babel.sea_disc_to_space(head)
5760         end
5761     end,
5762     'Babel.hyphenate')
5763 }}
5764 \endgroup
5765 %
5766 \def\bbl@provide@intraspace{%
5767 \bbl@ifunset\bbl@intsp@\languagename}{}%
5768 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5769 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5770 \ifin@ % cjk
5771 \bbl@cjk@intraspace
5772 \directlua{
5773     Babel.locale_props = Babel.locale_props or {}
5774     Babel.locale_props[\the\localeid].linebreak = 'c'
5775 }%
5776 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5777 \ifx\bbl@KVP@intrapenalty\@nnil
5778 \bbl@intrapenalty0\@%
5779 \fi
5780 \else % sea
5781 \bbl@sea@intraspace
5782 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5783 \directlua{
5784     Babel.sea_ranges = Babel.sea_ranges or {}

```

```

5785         Babel.set_chranges('\bbl@cl{sbcpr}',
5786                             '\bbl@cl{chrng}')
5787     }%
5788     \ifx\bbl@KVP@intrapenalty\@nnil
5789         \bbl@intrapenalty0\@@
5790     \fi
5791 \fi
5792 \fi
5793 \ifx\bbl@KVP@intrapenalty\@nnil\else
5794     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5795 \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.

```

5796 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5797 \def\bblar@chars{%
5798     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5799     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5800     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5801 \def\bblar@elongated{%
5802     0626,0628,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5803     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5804     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5805 \begingroup
5806     \catcode`\_ =11 \catcode`\:=11
5807     \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5808 \endgroup
5809 \gdef\bbl@arabicjust{%
5810     \let\bbl@arabicjust\relax
5811     \newattribute\bblar@kashida
5812     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bbblar@kashida' }%
5813     \bblar@kashida=\z@
5814     \bbl@patchfont{\bbl@parsejalt}}%
5815 \directlua{
5816     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5817     Babel.arabic.elong_map[\the\localeid] = {}
5818     luatexbase.add_to_callback('post_linebreak_filter',
5819         Babel.arabic.justify, 'Babel.arabic.justify')
5820     luatexbase.add_to_callback('hpack_filter',
5821         Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5822 }}%

```

Save both node lists to make replacement.

```

5823 \def\bblar@fetchjalt#1#2#3#4{%
5824     \bbl@exp{\bbl@foreach{#1}}{%
5825         \bbl@ifunset{bbblar@JE@##1}%
5826         {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5827         {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse{bbblar@JE@##1}#2}}%
5828     \directlua{%
5829         local last = nil
5830         for item in node.traverse(tex.box[0].head) do
5831             if item.id == node.id'glyph' and item.char > 0x600 and
5832             not (item.char == 0x200D) then
5833                 last = item
5834             end
5835         end
5836         Babel.arabic.#3['##1#4'] = last.char
5837     }}%

```

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

```

5838 \gdef\bbl@parsejalt{%
5839   \ifx\addfontfeature\undefined\else
5840     \bbl@xin@{/e}/{/bbl@cl{lbrk}}}%
5841   \ifin@
5842     \directlua{%
5843       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5844         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5845         tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5846       end
5847     }%
5848   \fi
5849 \fi}
5850 \gdef\bbl@parsejalti{%
5851   \begingroup
5852     \let\bbl@parsejalt\relax % To avoid infinite loop
5853     \edef\bbl@tempb{\fontid\font}%
5854     \bblar@nofswarn
5855     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5856     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5857     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5858     \addfontfeature{RawFeature+=jalt}%
5859     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5860     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5861     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5862     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5863     \directlua{%
5864       for k, v in pairs(Babel.arabic.from) do
5865         if Babel.arabic.dest[k] and
5866           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5867           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5868             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5869         end
5870       end
5871     }%
5872   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5873 \begingroup
5874 \catcode`#=11
5875 \catcode`~=11
5876 \directlua{
5877
5878 Babel.arabic = Babel.arabic or {}
5879 Babel.arabic.from = {}
5880 Babel.arabic.dest = {}
5881 Babel.arabic.justify_factor = 0.95
5882 Babel.arabic.justify_enabled = true
5883 Babel.arabic.kashida_limit = -1
5884
5885 function Babel.arabic.justify(head)
5886   if not Babel.arabic.justify_enabled then return head end
5887   for line in node.traverse_id(node.id'hlist', head) do
5888     Babel.arabic.justify_hlist(head, line)
5889   end
5890   % In case the very first item is a line (eg, in \vbox):
5891   while head.prev do head = head.prev end
5892   return head
5893 end
5894
5895 function Babel.arabic.justify_hbox(head, gc, size, pack)
5896   local has_inf = false
5897   if Babel.arabic.justify_enabled and pack == 'exactly' then
5898     for n in node.traverse_id(12, head) do

```

```

5899     if n.stretch_order > 0 then has_inf = true end
5900   end
5901   if not has_inf then
5902     Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5903   end
5904 end
5905 return head
5906 end
5907
5908 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5909   local d, new
5910   local k_list, k_item, pos_inline
5911   local width, width_new, full, k_curr, wt_pos, goal, shift
5912   local subst_done = false
5913   local elong_map = Babel.arabic.elong_map
5914   local cnt
5915   local last_line
5916   local GLYPH = node.id'glyph'
5917   local KASHIDA = Babel.attr_kashida
5918   local LOCALE = Babel.attr_locale
5919
5920   if line == nil then
5921     line = {}
5922     line.glue_sign = 1
5923     line.glue_order = 0
5924     line.head = head
5925     line.shift = 0
5926     line.width = size
5927   end
5928
5929   % Exclude last line. todo. But-- it discards one-word lines, too!
5930   % ? Look for glue = 12:15
5931   if (line.glue_sign == 1 and line.glue_order == 0) then
5932     elongs = {}      % Stores elongated candidates of each line
5933     k_list = {}      % And all letters with kashida
5934     pos_inline = 0   % Not yet used
5935
5936     for n in node.traverse_id(GLYPH, line.head) do
5937       pos_inline = pos_inline + 1 % To find where it is. Not used.
5938
5939       % Elongated glyphs
5940       if elong_map then
5941         local locale = node.get_attribute(n, LOCALE)
5942         if elong_map[locale] and elong_map[locale][n.font] and
5943           elong_map[locale][n.font][n.char] then
5944           table.insert(elongs, {node = n, locale = locale} )
5945           node.set_attribute(n.prev, KASHIDA, 0)
5946         end
5947       end
5948
5949       % Tatwil. First create a list of nodes marked with kashida. The
5950       % rest of nodes can be ignored. The list of used weights is build
5951       % when transforms with the key kashida= are declared.
5952       if Babel.kashida_wts then
5953         local k_wt = node.get_attribute(n, KASHIDA)
5954         if k_wt > 0 then % todo. parameter for multi inserts
5955           table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5956         end
5957       end
5958
5959     end % of node.traverse_id
5960
5961     if #elongs == 0 and #k_list == 0 then goto next_line end

```

```

5962 full = line.width
5963 shift = line.shift
5964 goal = full * Babel.arabic.justify_factor % A bit crude
5965 width = node.dimensions(line.head) % The 'natural' width
5966
5967 % == Elongated ==
5968 % Original idea taken from 'chickenize'
5969 while (#elongs > 0 and width < goal) do
5970     subst_done = true
5971     local x = #elongs
5972     local curr = elongs[x].node
5973     local oldchar = curr.char
5974     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5975     width = node.dimensions(line.head) % Check if the line is too wide
5976     % Substitute back if the line would be too wide and break:
5977     if width > goal then
5978         curr.char = oldchar
5979         break
5980     end
5981     % If continue, pop the just substituted node from the list:
5982     table.remove(elongs, x)
5983 end
5984
5985 % == Tatwil ==
5986 % Traverse the kashida node list so many times as required, until
5987 % the line is filled. The first pass adds a tatweel after each
5988 % node with kashida in the line, the second pass adds another one,
5989 % and so on. In each pass, add first the kashida with the highest
5990 % weight, then with lower weight and so on.
5991 if #k_list == 0 then goto next_line end
5992
5993 width = node.dimensions(line.head) % The 'natural' width
5994 k_curr = #k_list % Traverse backwards, from the end
5995 wt_pos = 1
5996
5997 while width < goal do
5998     subst_done = true
5999     k_item = k_list[k_curr].node
6000     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6001         d = node.copy(k_item)
6002         d.char = 0x0640
6003         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6004         d.xoffset = 0
6005         line.head, new = node.insert_after(line.head, k_item, d)
6006         width_new = node.dimensions(line.head)
6007         if width > goal or width == width_new then
6008             node.remove(line.head, new) % Better compute before
6009             break
6010         end
6011         if Babel.fix_diacr then
6012             Babel.fix_diacr(k_item.next)
6013         end
6014         width = width_new
6015     end
6016     if k_curr == 1 then
6017         k_curr = #k_list
6018         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6019     else
6020         k_curr = k_curr - 1
6021     end
6022 end
6023
6024 % Limit the number of tatweel by removing them. Not very efficient,

```

```

6025 % but it does the job in a quite predictable way.
6026 if Babel.arabic.kashida_limit > -1 then
6027   cnt = 0
6028   for n in node.traverse_id(GLYPH, line.head) do
6029     if n.char == 0x0640 then
6030       cnt = cnt + 1
6031       if cnt > Babel.arabic.kashida_limit then
6032         node.remove(line.head, n)
6033       end
6034     else
6035       cnt = 0
6036     end
6037   end
6038 end
6039
6040 ::next_line::
6041
6042 % Must take into account marks and ins, see luatex manual.
6043 % Have to be executed only if there are changes. Investigate
6044 % what's going on exactly.
6045 if subst_done and not gc then
6046   d = node.hpack(line.head, full, 'exactly')
6047   d.shift = shift
6048   node.insert_before(head, line, d)
6049   node.remove(head, line)
6050 end
6051 end % if process line
6052 end
6053 }
6054 \endgroup
6055 \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`.

```

6056 \def\bbl@scr@node@list{%
6057   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6058   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6059 \ifnum\bbl@bidimode=102 % bidi-r
6060   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6061 \fi
6062 \def\bbl@set@renderer{%
6063   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6064   \ifin@
6065     \let\bbl@unset@renderer\relax
6066   \else
6067     \bbl@exp{%
6068       \def\\bbl@unset@renderer{%
6069         \def<g__fontspec_default_fontopts_clist>{%
6070           \[g__fontspec_default_fontopts_clist]}%
6071         \def<g__fontspec_default_fontopts_clist>{%
6072           Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}%
6073       \fi}
6074 <@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.

```

6075 \directlua{% DL6
6076 Babel.script_blocks = {
6077   ['dflt'] = {},
6078   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6079               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6080   ['Armn'] = {{0x0530, 0x058F}},
6081   ['Beng'] = {{0x0980, 0x09FF}},
6082   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6083   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6084   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6085              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6086   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6087   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6088              {0xAB00, 0xAB2F}},
6089   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6090   % Don't follow strictly Unicode, which places some Coptic letters in
6091   % the 'Greek and Coptic' block
6092   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6093   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6094              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6095              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6096              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6097              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6098              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6099   ['Hebr'] = {{0x0590, 0x05FF},
6100              {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6101   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6102              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6103   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6104   ['Knda'] = {{0x0C80, 0x0CFF}},
6105   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6106              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6107              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6108   ['Lao'] = {{0x0E80, 0x0EFF}},
6109   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6110              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6111              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6112   ['Mahj'] = {{0x11150, 0x1117F}},
6113   ['Mlym'] = {{0x0D00, 0x0D7F}},
6114   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6115   ['Orya'] = {{0x0B00, 0x0B7F}},
6116   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6117   ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6118   ['Taml'] = {{0x0B80, 0x0BFF}},
6119   ['Telu'] = {{0x0C00, 0x0C7F}},
6120   ['Tfng'] = {{0x2D30, 0x2D7F}},
6121   ['Thai'] = {{0x0E00, 0x0E7F}},
6122   ['Tibt'] = {{0x0F00, 0x0FFF}},
6123   ['Vaii'] = {{0xA500, 0xA63F}},
6124   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6125 }
6126
6127 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6128 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6129 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6130
6131 function Babel.locale_map(head)

```

```

6132 if not Babel.locale_mapped then return head end
6133
6134 local LOCALE = Babel.attr_locale
6135 local GLYPH = node.id('glyph')
6136 local inmath = false
6137 local toloc_save
6138 for item in node.traverse(head) do
6139     local toloc
6140     if not inmath and item.id == GLYPH then
6141         % Optimization: build a table with the chars found
6142         if Babel.chr_to_loc[item.char] then
6143             toloc = Babel.chr_to_loc[item.char]
6144         else
6145             for lc, maps in pairs(Babel.loc_to_scr) do
6146                 for _, rg in pairs(maps) do
6147                     if item.char >= rg[1] and item.char <= rg[2] then
6148                         Babel.chr_to_loc[item.char] = lc
6149                         toloc = lc
6150                         break
6151                     end
6152                 end
6153             end
6154             % Treat composite chars in a different fashion, because they
6155             % 'inherit' the previous locale.
6156             if (item.char >= 0x0300 and item.char <= 0x036F) or
6157                 (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6158                 (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6159                 Babel.chr_to_loc[item.char] = -2000
6160                 toloc = -2000
6161             end
6162             if not toloc then
6163                 Babel.chr_to_loc[item.char] = -1000
6164             end
6165         end
6166         if toloc == -2000 then
6167             toloc = toloc_save
6168         elseif toloc == -1000 then
6169             toloc = nil
6170         end
6171         if toloc and Babel.locale_props[toloc] and
6172             Babel.locale_props[toloc].letters and
6173             tex.getcatcode(item.char) \string~= 11 then
6174             toloc = nil
6175         end
6176         if toloc and Babel.locale_props[toloc].script
6177             and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6178             and Babel.locale_props[toloc].script ==
6179             Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6180             toloc = nil
6181         end
6182         if toloc then
6183             if Babel.locale_props[toloc].lg then
6184                 item.lang = Babel.locale_props[toloc].lg
6185                 node.set_attribute(item, LOCALE, toloc)
6186             end
6187             if Babel.locale_props[toloc]['/'..item.font] then
6188                 item.font = Babel.locale_props[toloc]['/'..item.font]
6189             end
6190         end
6191         toloc_save = toloc
6192     elseif not inmath and item.id == 7 then % Apply recursively
6193         item.replace = item.replace and Babel.locale_map(item.replace)
6194         item.pre = item.pre and Babel.locale_map(item.pre)

```

```

6195     item.post    = item.post and Babel.locale_map(item.post)
6196     elseif item.id == node.id'math' then
6197         inmath = (item.subtype == 0)
6198     end
6199 end
6200 return head
6201 end
6202 }

```

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

```

6203 \newcommand\babelcharproperty[1]{%
6204   \count@=#1\relax
6205   \ifvmode
6206     \expandafter\bbl@chprop
6207   \else
6208     \bbl@error{charproperty-only-vertical}{#1}%
6209   \fi}
6210 \newcommand\bbl@chprop[3][\the\count@]{%
6211   \@tempcnta=#1\relax
6212   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6213   {\bbl@error{unknown-char-property}{#2}}%
6214   {%
6215     \loop
6216     \bbl@cs{chprop@#2}{#3}%
6217     \ifnum\count@<\@tempcnta
6218       \advance\count@\@ne
6219     \repeat}
6220 %
6221 \def\bbl@chprop@direction#1{%
6222   \directlua{
6223     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6224     Babel.characters[\the\count@]['d'] = '#1'
6225   }}
6226 \let\bbl@chprop@bc\bbl@chprop@direction
6227 %
6228 \def\bbl@chprop@mirror#1{%
6229   \directlua{
6230     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6231     Babel.characters[\the\count@]['m'] = '\number#1'
6232   }}
6233 \let\bbl@chprop@bmg\bbl@chprop@mirror
6234 %
6235 \def\bbl@chprop@linebreak#1{%
6236   \directlua{
6237     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6238     Babel.cjk_characters[\the\count@]['c'] = '#1'
6239   }}
6240 \let\bbl@chprop@lb\bbl@chprop@linebreak
6241 %
6242 \def\bbl@chprop@locale#1{%
6243   \directlua{
6244     Babel.chr_to_loc = Babel.chr_to_loc or {}
6245     Babel.chr_to_loc[\the\count@] =
6246       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6247   }}

```

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.

```

6248 \directlua{% DL7
6249   Babel.nohyphenation = \the\l@nohyphenation
6250 }

```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the `{n}` syntax. For example, `pre={1}{1}-`

becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to `lua load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

6251 \begingroup
6252 \catcode\~ = 12
6253 \catcode\% = 12
6254 \catcode\& = 14
6255 \catcode\| = 12
6256 \gdef\babelprehyphenation{%&
6257   \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6258 \gdef\babelposthyphenation{%&
6259   \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6260 %
6261 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6262   \ifcase#1
6263     \bbl@activateprehyphen
6264   \or
6265     \bbl@activateposthyphen
6266   \fi
6267 \begingroup
6268   \def\babeltempa{\bbl@add@list\babeltempb}%&
6269   \let\babeltempb\empty
6270   \def\bbl@tempa{#5}%&
6271   \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6272   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6273     \bbl@ifsamestring{##1}{remove}%&
6274     {\bbl@add@list\babeltempb{nil}}}%&
6275   {\directlua{
6276     local rep = [=##1]=]
6277     local three_args = '%s*=%s*([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)%s+([%-d%.%a{}|]+)'
6278     & Numeric passes directly: kern, penalty...
6279     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6280     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6281     rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6282     rep = rep:gsub('(string)%s*=%s*([%^s,]*)', Babel.capture_func)
6283     rep = rep:gsub('node%s*=%s*([%a+)%s*([%a+])', Babel.capture_node)
6284     rep = rep:gsub(' (norule)' .. three_args,
6285       'norule = {' .. '%2, %3, %4' .. '}')
6286     if #1 == 0 or #1 == 2 then
6287       rep = rep:gsub(' (space)' .. three_args,
6288         'space = {' .. '%2, %3, %4' .. '}')
6289       rep = rep:gsub(' (spacefactor)' .. three_args,
6290         'spacefactor = {' .. '%2, %3, %4' .. '}')
6291       rep = rep:gsub(' (kashida)%s*=%s*([%^s,]*)', Babel.capture_kashida)
6292       & Transform values
6293       rep, n = rep:gsub(' {([%a%-%.]+)|([%a%_%.]+)}',
6294         function(v,d)
6295           return string.format (
6296             '{\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6297             v,
6298             load( 'return Babel.locale_props'..
6299               '[\the\csname bbl@id@@#3\endcsname].' .. d)() )
6300           end )
6301       rep, n = rep:gsub(' {([%a%-%.]+)|([%-d%.]+)}',
6302         '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6303     end
6304     if #1 == 1 then
6305       rep = rep:gsub(' (no)%s*=%s*([%^s,]*)', Babel.capture_func)
6306       rep = rep:gsub(' (pre)%s*=%s*([%^s,]*)', Babel.capture_func)

```

```

6307         rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6308     end
6309     tex.print([[\\string\\babeltempa{[]] .. rep .. [[]]])
6310 }}&%
6311 \\bbl@foreach\\babeltempb{&%
6312     \\bbl@forkv{##1}}{&%
6313         \\in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6314             post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6315         \\ifin@\\else
6316             \\bbl@error{bad-transform-option}{###1}{}}{&%
6317         \\fi}}&%
6318 \\let\\bbl@kv@attribute\\relax
6319 \\let\\bbl@kv@label\\relax
6320 \\let\\bbl@kv@fonts@empty
6321 \\let\\bbl@kv@prepend\\relax
6322 \\bbl@forkv{#2}{\\bbl@csarg\\edef{kv##1}{##2}}&%
6323 \\ifx\\bbl@kv@fonts@empty\\else\\bbl@settransfont\\fi
6324 \\ifx\\bbl@kv@attribute\\relax
6325     \\ifx\\bbl@kv@label\\relax\\else
6326         \\bbl@exp{\\bbl@trim@def\\bbl@kv@fonts{\\bbl@kv@fonts}}&%
6327         \\bbl@replace\\bbl@kv@fonts{ }{,}&%
6328         \\edef\\bbl@kv@attribute{\\bbl@ATR@\\bbl@kv@label @#3@\\bbl@kv@fonts}&%
6329         \\count@\\z@
6330         \\def\\bbl@elt##1##2##3{&%
6331             \\bbl@ifsamestring{#3,\\bbl@kv@label}{##1,##2}&%
6332             {\\bbl@ifsamestring{\\bbl@kv@fonts}{##3}&%
6333                 {\\count@\\@ne}&%
6334                 {\\bbl@error{font-conflict-transforms}{}}{}}}&%
6335             }&%
6336         \\bbl@transfont@list
6337         \\ifnum\\count@=\\z@
6338             \\bbl@exp{\\global\\bbl@add\\bbl@transfont@list
6339                 {\\bbl@elt{#3}{\\bbl@kv@label}{\\bbl@kv@fonts}}}&%
6340             \\fi
6341             \\bbl@ifunset{\\bbl@kv@attribute}&%
6342             {\\global\\bbl@carg\\newattribute{\\bbl@kv@attribute}}&%
6343             {}&%
6344             \\global\\bbl@carg\\setattribute{\\bbl@kv@attribute}\\@ne
6345             \\fi
6346         \\else
6347             \\edef\\bbl@kv@attribute{\\expandafter\\bbl@stripslash\\bbl@kv@attribute}&%
6348             \\fi
6349         \\directlua{
6350             local lbkr = Babel.linebreaking.replacements[#1]
6351             local u = unicode.utf8
6352             local id, attr, label
6353             if #1 == 0 then
6354                 id = \\the\\csname bbl@id@@#3\\endcsname\\space
6355             else
6356                 id = \\the\\csname l@#3\\endcsname\\space
6357             end
6358             \\ifx\\bbl@kv@attribute\\relax
6359                 attr = -1
6360             \\else
6361                 attr = luatexbase.registernumber'\\bbl@kv@attribute'
6362             \\fi
6363             \\ifx\\bbl@kv@label\\relax\\else &% Same refs:
6364                 label = [==[\\bbl@kv@label]==]
6365             \\fi
6366             &% Convert pattern:
6367             local patt = string.gsub([==[#4]==], '%s', '')
6368             if #1 == 0 then
6369                 patt = string.gsub(patt, '|', ' ')

```

```

6370     end
6371     if not u.find(patt, '()', nil, true) then
6372         patt = '()' .. patt .. '()'
6373     end
6374     if #l == 1 then
6375         patt = string.gsub(patt, '%(%)^', '^()')
6376         patt = string.gsub(patt, '%$(%)', '()$')
6377     end
6378     patt = u.gsub(patt, '{(.)}',
6379         function (n)
6380             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6381         end)
6382     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6383         function (n)
6384             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6385         end)
6386     lbkr[id] = lbkr[id] or {}
6387     table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6388         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6389 }&%
6390 \endgroup}
6391 \endgroup
6392 %
6393 \let\bbl@transfont@list@empty
6394 \def\bbl@settransfont{%
6395     \global\let\bbl@settransfont\relax % Execute only once
6396     \gdef\bbl@transfont{%
6397         \def\bbl@elt####1####2####3{%
6398             \bbl@ifblank{####3}%
6399                 {\count@tw@}% Do nothing if no fonts
6400                 {\count@z@
6401                     \bbl@vforeach{####3}{%
6402                         \def\bbl@tempd{#####1}%
6403                         \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6404                         \ifx\bbl@tempd\bbl@tempe
6405                             \count@ne
6406                         \else\ifx\bbl@tempd\bbl@transfam
6407                             \count@ne
6408                         \fi\fi}%
6409                     \ifcase\count@
6410                         \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6411                     \or
6412                         \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6413                     \fi}}%
6414                 \bbl@transfont@list}%
6415     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6416     \gdef\bbl@transfam{-unknown-}%
6417     \bbl@foreach\bbl@font@fams{%
6418         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6419         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6420         {\xdef\bbl@transfam{##1}}%
6421     }}}}
6422 %
6423 \DeclareRobustCommand\enablelocaletransform[1]{%
6424     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6425     {\bbl@error{transform-not-available}{#1}}}%
6426     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6427 \DeclareRobustCommand\disablelocaletransform[1]{%
6428     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6429     {\bbl@error{transform-not-available-b}{#1}}}%
6430     {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}

```

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

add_after and add_before.

```

6431 \def\bbl@activateposthyphen{%
6432   \let\bbl@activateposthyphen\relax
6433   \ifx\bbl@attr@hboxed\@undefined
6434     \newattribute\bbl@attr@hboxed
6435   \fi
6436   \directlua{
6437     require('babel-transforms.lua')
6438     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6439   }}
6440 \def\bbl@activateprehyphen{%
6441   \let\bbl@activateprehyphen\relax
6442   \ifx\bbl@attr@hboxed\@undefined
6443     \newattribute\bbl@attr@hboxed
6444   \fi
6445   \directlua{
6446     require('babel-transforms.lua')
6447     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6448   }}
6449 \newcommand\SetTransformValue[3]{%
6450   \directlua{
6451     Babel.locale_props[\the\csname bbl@id@@#1\endcsname].vars["#2"] = #3
6452   }}

```

The code in `babel-transforms.lua` prints at some points the current string being transformed. This macro first make sure this file is loaded. Then, activates temporarily this feature and typeset inside a box the text in the argument.

```

6453 \newcommand\ShowBabelTransforms[1]{%
6454   \bbl@activateprehyphen
6455   \bbl@activateposthyphen
6456   \begingroup
6457     \directlua{ Babel.show_transforms = true }%
6458     \setbox\z@\vbox{#1}%
6459     \directlua{ Babel.show_transforms = false }%
6460   \endgroup}

```

The following experimental (and unfinished) macro applies the prehyphenation transforms for the current locale to a string (characters and spaces) and processes it in a fully expandable way (among other limitations, the string can't contain `]==]`). The way it operates is admittedly rather cumbersome: it converts the string to a node list, processes it, and converts it back to a string. The lua code is in the lua file below.

```

6461 \newcommand\localeprehyphenation[1]{%
6462   \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 `ℒTEX`. Just in case, consider the possibility it has not been loaded.

```

6463 \def\bbl@activate@preotf{%
6464   \let\bbl@activate@preotf\relax % only once
6465   \directlua{
6466     function Babel.pre_otfload_v(head)
6467       if Babel.numbers and Babel.digits_mapped then
6468         head = Babel.numbers(head)
6469       end
6470       if Babel.bidi_enabled then
6471         head = Babel.bidi(head, false, dir)
6472       end
6473       return head
6474     end
6475     %
6476     function Babel.pre_otfload_h(head, gc, sz, pt, dir)

```

```

6477     if Babel.numbers and Babel.digits_mapped then
6478         head = Babel.numbers(head)
6479     end
6480     if Babel.bidi_enabled then
6481         head = Babel.bidi(head, false, dir)
6482     end
6483     return head
6484 end
6485 %
6486 luatexbase.add_to_callback('pre_linebreak_filter',
6487     Babel.pre_otfload_v,
6488     'Babel.pre_otfload_v',
6489     Babel.priority_in_callback('pre_linebreak_filter',
6490         'luaotfload.node_processor') or nil)
6491 %
6492 luatexbase.add_to_callback('hpack_filter',
6493     Babel.pre_otfload_h,
6494     'Babel.pre_otfload_h',
6495     Babel.priority_in_callback('hpack_filter',
6496         'luaotfload.node_processor') or nil)
6497 }}

```

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`.

```

6498 \breakafterdirmode=1
6499 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6500 \let\bbl@beforeforeign\leavevmode
6501 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6502 \RequirePackage{luatexbase}
6503 \bbl@activate@preotf
6504 \directlua{
6505     require('babel-data-bidi.lua')
6506     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6507         require('babel-bidi-basic.lua')
6508     \or
6509         require('babel-bidi-basic-r.lua')
6510     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6511     table.insert(Babel.ranges, {0xF0000, 0xFFFFD, 'on'})
6512     table.insert(Babel.ranges, {0x100000, 0x10FFFFD, 'on'})
6513 \fi}
6514 \newattribute\bbl@attr@dir
6515 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6516 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6517 \fi
6518 %
6519 \chardef\bbl@thetextdir\z@
6520 \chardef\bbl@thepardir\z@
6521 \def\bbl@getluadir#1{%
6522     \directlua{
6523         if tex.#ldir == 'TLT' then
6524             tex.sprint('0')
6525         elseif tex.#ldir == 'TRT' then
6526             tex.sprint('1')
6527         else
6528             tex.sprint('0')
6529         end}}
6530 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6531     \ifcase#3\relax
6532     \ifcase\bbl@getluadir{#1}\relax\else
6533         #2 TLT\relax
6534     \fi

```



```

6535 \else
6536   \ifcase\bb@getluadir{#1}\relax
6537     #2 TRT\relax
6538   \fi
6539 \fi}

```

\bb@attr@dir stores the directions with a mask: ..00PPTT, with masks 0xC (PP is the par dir) and 0x3 (TT is the text dir).

```

6540 \def\bb@thedir{0}
6541 \def\bb@textdir#1{%
6542   \bb@setluadir{text}\textdir{#1}%
6543   \chardef\bb@thetextdir#1\relax
6544   \edef\bb@thedir{\the\numexpr\bb@thepardir*4+#1}%
6545   \setattribute\bb@attr@dir{\numexpr\bb@thepardir*4+#1}}
6546 \def\bb@pardir#1{% Used twice
6547   \bb@setluadir{par}\pardir{#1}%
6548   \chardef\bb@thepardir#1\relax}
6549 \def\bb@bodydir{\bb@setluadir{body}\bodydir}% Used once
6550 \def\bb@pagedir{\bb@setluadir{page}\pagedir}% Unused
6551 \def\bb@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.

```

6552 \ifnum\bb@bidimode>\z@ % Any bidi=
6553   \def\bb@insidemath{0}%
6554   \def\bb@everymath{\def\bb@insidemath{1}}
6555   \def\bb@everydisplay{\def\bb@insidemath{2}}
6556   \frozen@everymath\expandafter{%
6557     \expandafter\bb@everymath\the\frozen@everymath}
6558   \frozen@everydisplay\expandafter{%
6559     \expandafter\bb@everydisplay\the\frozen@everydisplay}
6560   \AtBeginDocument{
6561     \directlua{
6562       function Babel.math_box_dir(head)
6563         if not (token.get_macro('bb@insidemath') == '0') then
6564           if Babel.hlist_has_bidi(head) then
6565             local d = node.new(node.id'dir')
6566             d.dir = '+TRT'
6567             node.insert_before(head, node.has_glyph(head), d)
6568             local inmath = false
6569             for item in node.traverse(head) do
6570               if item.id == 11 then
6571                 inmath = (item.subtype == 0)
6572               elseif not inmath then
6573                 node.set_attribute(item,
6574                   Babel.attr_dir, token.get_macro('bb@thedir'))
6575             end
6576           end
6577         end
6578       end
6579       return head
6580     end
6581     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6582       "Babel.math_box_dir", 0)
6583     if Babel.unset_atdir then
6584       luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6585         "Babel.unset_atdir")
6586       luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6587         "Babel.unset_atdir")
6588     end
6589   }}%
6590 \fi

```

Experimental. Tentative name.

```

6591 \DeclareRobustCommand\localebox[1]{%
6592   {\def\bbl@insidemath{0}%
6593    \mbox{\foreignlanguage{\language}\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.

```

6594 \bbl@trace{Redefinitions for bidi layout}
6595 %
6596 <<{*More package options}>> ≡
6597 \chardef\bbl@eqnpos\z@
6598 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6599 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6600 <</More package options>>
6601 %
6602 \ifnum\bbl@bidimode>\z@ % Any bidi=
6603   \matheqdirmode\@ne % A luatex primitive
6604   \let\bbl@eqnodir\relax
6605   \def\bbl@eqdel{()}
6606   \def\bbl@eqnum{%
6607     {\normalfont\normalcolor
6608      \expandafter\@firstoftwo\bbl@eqdel
6609      \theequation
6610      \expandafter\@secondoftwo\bbl@eqdel}}
6611   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6612   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6613   \def\bbl@eqno@flip#1{%
6614     \ifdim\predisplaysize=-\maxdimen
6615       \eqno
6616       \hb@xt@.01pt{%
6617         \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}\hss}%
6618       \else
6619         \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6620     \fi
6621     \bbl@exp{\def\\@currentlabel{\bbl@upset}}}}
6622   \def\bbl@leqno@flip#1{%
6623     \ifdim\predisplaysize=-\maxdimen
6624       \leqno
6625       \hb@xt@.01pt{%
6626         \hss\hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}\hss}%
6627     \else
6628       \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6629     \fi
6630     \bbl@exp{\def\\@currentlabel{\bbl@upset}}}}
6631 %

```

```

6632 \AtBeginDocument{%
6633 \ifx\bbbl@noamsmath\relax\else
6634 \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6635 \AddToHook{env/equation/begin}{%
6636 \ifnum\bbbl@thetextdir>\z@
6637 \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6638 \let\@eqnnum\bbbl@eqnum
6639 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6640 \chardef\bbbl@thetextdir\z@
6641 \bbbl@add\normalfont{\bbbl@eqnodir}%
6642 \ifcase\bbbl@eqnpos
6643 \let\bbbl@puteqno\bbbl@eqno@flip
6644 \or
6645 \let\bbbl@puteqno\bbbl@leqno@flip
6646 \fi
6647 \fi}%
6648 \ifnum\bbbl@eqnpos=\tw@\else
6649 \def\endequation{\bbbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6650 \fi
6651 \AddToHook{env/eqnarray/begin}{%
6652 \ifnum\bbbl@thetextdir>\z@
6653 \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6654 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6655 \chardef\bbbl@thetextdir\z@
6656 \bbbl@add\normalfont{\bbbl@eqnodir}%
6657 \ifnum\bbbl@eqnpos=\@ne
6658 \def\@eqnnum{%
6659 \setbox\z@\hbox{\bbbl@eqnum}%
6660 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6661 \else
6662 \let\@eqnnum\bbbl@eqnum
6663 \fi
6664 \fi}
6665 % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6666 \expandafter\bbbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$}$%
6667 \else % amstex
6668 \bbbl@exp{% Hack to hide maybe undefined conditionals:
6669 \chardef\bbbl@eqnpos=0%
6670 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6671 \ifnum\bbbl@eqnpos=\@ne
6672 \let\bbbl@ams@lap\hbox
6673 \else
6674 \let\bbbl@ams@lap\llap
6675 \fi
6676 \ExplSyntaxOn % Required by \bbbl@sreplace with \intertext@
6677 \bbbl@sreplace\intertext@{\normalbaselines}%
6678 {\normalbaselines
6679 \ifx\bbbl@eqnodir\relax\else\bbbl@paddir\@ne\bbbl@eqnodir\fi}%
6680 \ExplSyntaxOff
6681 \def\bbbl@ams@tagbox#1#2{#1{\bbbl@eqnodir#2}}% #1=hbox|@lap|flip
6682 \ifx\bbbl@ams@lap\hbox % leqno
6683 \def\bbbl@ams@flip#1{%
6684 \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6685 \else % eqno
6686 \def\bbbl@ams@flip#1{%
6687 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6688 \fi
6689 \def\bbbl@ams@preset#1{%
6690 \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6691 \ifnum\bbbl@thetextdir>\z@
6692 \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6693 \bbbl@sreplace\textdef@{\hbox}{\bbbl@ams@tagbox\hbox}%
6694 \bbbl@sreplace\maketag@@@{\hbox}{\bbbl@ams@tagbox#1}%

```

```

6695 \fi}%
6696 \ifnum\bbl@eqnpos=\tw@\else
6697 \def\bbl@ams@equation{%
6698 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6699 \ifnum\bbl@thetextdir>\z@
6700 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6701 \chardef\bbl@thetextdir\z@
6702 \bbl@add\normalfont{\bbl@eqnodir}%
6703 \ifcase\bbl@eqnpos
6704 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6705 \or
6706 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6707 \fi
6708 \fi}%
6709 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6710 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6711 \fi
6712 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6713 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6714 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6715 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6716 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6717 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6718 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6719 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6720 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6721 % Hackish, for proper alignment. Don't ask me why it works!:
6722 \bbl@exp{% Avoid a 'visible' conditional
6723 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6724 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6725 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6726 \AddToHook{env/split/before}{%
6727 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6728 \ifnum\bbl@thetextdir>\z@
6729 \bbl@ifsamestring\@currenvir{equation}%
6730 {\ifx\bbl@ams@lap\hbox % leqno
6731 \def\bbl@ams@flip#1{%
6732 \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6733 \else
6734 \def\bbl@ams@flip#1{%
6735 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss#1}}}%
6736 \fi}%
6737 }%
6738 \fi}%
6739 \fi\fi}
6740 \fi

```

Declarations specific to lua, called by \babelprovide.

```

6741 \def\bbl@provide@extra#1{%
6742 % == onchar ==
6743 \ifx\bbl@KVP@onchar\@nnil\else
6744 \bbl@luahyphenate
6745 \bbl@exp{%
6746 \\\AddToHook{env/document/before}{\select@language{#1}}}%
6747 \directlua{
6748 if Babel.locale_mapped == nil then
6749 Babel.locale_mapped = true
6750 Babel.linebreaking.add_before(Babel.locale_map, 1)
6751 Babel.loc_to_scr = {}
6752 Babel.chr_to_loc = Babel.chr_to_loc or {}
6753 end
6754 Babel.locale_props[\the\localeid].letters = false
6755 }%

```

```

6756 \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6757 \ifin@
6758 \directlua{
6759     Babel.locale_props[\the\localeid].letters = true
6760 }%
6761 \fi
6762 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6763 \ifin@
6764 \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6765     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}}%
6766 \fi
6767 \bbl@exp{\bbl@add\bbl@starthyphens
6768     {\bbl@patterns@lua{\languageName}}}%
6769 \directlua{
6770     if Babel.script_blocks['\bbl@cl{sbcP}'] then
6771         Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcP}']
6772         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@languageName}\space
6773     end
6774 }%
6775 \fi
6776 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6777 \ifin@
6778     \bbl@ifunset{bbl@lsys@languageName}{\bbl@provide@lsys@languageName}}}%
6779     \bbl@ifunset{bbl@wdir@languageName}{\bbl@provide@dirs@languageName}}}%
6780 \directlua{
6781     if Babel.script_blocks['\bbl@cl{sbcP}'] then
6782         Babel.loc_to_scr[\the\localeid] =
6783             Babel.script_blocks['\bbl@cl{sbcP}']
6784     end}%
6785 \ifx\bbl@mapselect\@undefined
6786     \AtBeginDocument{%
6787         \bbl@patchfont{\bbl@mapselect}}%
6788         {\selectfont}}%
6789     \def\bbl@mapselect{%
6790         \let\bbl@mapselect\relax
6791         \edef\bbl@prefontid{\fontid\font}}%
6792     \def\bbl@mapdir##1{%
6793         \begingroup
6794             \setbox\z@\hbox{% Force text mode
6795                 \def\languageName{##1}%
6796                 \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6797                 \bbl@switchfont
6798                 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6799                     \directlua{
6800                         Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6801                             ['/\bbl@prefontid'] = \fontid\font\space}%
6802                     \fi}%
6803             \endgroup}%
6804     \fi
6805     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{languageName}}}%
6806 \fi
6807 \fi
6808 % == mapfont ==
6809 % For bidi texts, to switch the font based on direction. Deprecated
6810 \ifx\bbl@KVP@mapfont\@nnil\else
6811     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
6812     {\bbl@error{unknown-mapfont}}}%
6813     \bbl@ifunset{bbl@lsys@languageName}{\bbl@provide@lsys@languageName}}}%
6814     \bbl@ifunset{bbl@wdir@languageName}{\bbl@provide@dirs@languageName}}}%
6815 \ifx\bbl@mapselect\@undefined
6816     \AtBeginDocument{%
6817         \bbl@patchfont{\bbl@mapselect}}%
6818         {\selectfont}}%

```

```

6819 \def\bbl@mapselect{%
6820 \let\bbl@mapselect\relax
6821 \edef\bbl@prefontid{\fontid\font}}%
6822 \def\bbl@mapdir##1{%
6823 {\def\language{##1}%
6824 \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6825 \bbl@switchfont
6826 \directlua{Babel.fontmap
6827 [\the\csname bbl@wdir@##1\endcsname]%
6828 [\bbl@prefontid]=\fontid\font}}}%
6829 \fi
6830 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
6831 \fi
6832 % == Line breaking: CJK quotes ==
6833 \ifcase\bbl@engine\or
6834 \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}%
6835 \ifin@
6836 \bbl@ifunset{bbl@quote@\language}{}%
6837 {\directlua{
6838 Babel.locale_props[\the\localeid].cjk_quotes = {}
6839 local cs = 'op'
6840 for c in string.utfvalues(%
6841 [[\csname bbl@quote@\language\endcsname]]) do
6842 if Babel.cjk_characters[c].c == 'qu' then
6843 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6844 end
6845 cs = ( cs == 'op') and 'cl' or 'op'
6846 end
6847 }}%
6848 \fi
6849 \fi
6850 % == Counters: mapdigits ==
6851 % Native digits
6852 \ifx\bbl@KVP@mapdigits\@nnil\else
6853 \bbl@ifunset{bbl@dgnat@\language}{}%
6854 {\bbl@activate@preotf
6855 \directlua{
6856 Babel.digits_mapped = true
6857 Babel.digits = Babel.digits or {}
6858 Babel.digits[\the\localeid] =
6859 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6860 if not Babel.numbers then
6861 function Babel.numbers(head)
6862 local LOCALE = Babel.attr_locale
6863 local GLYPH = node.id'glyph'
6864 local inmath = false
6865 for item in node.traverse(head) do
6866 if not inmath and item.id == GLYPH then
6867 local temp = node.get_attribute(item, LOCALE)
6868 if Babel.digits[temp] then
6869 local chr = item.char
6870 if chr > 47 and chr < 58 then
6871 item.char = Babel.digits[temp][chr-47]
6872 end
6873 end
6874 elseif item.id == node.id'math' then
6875 inmath = (item.subtype == 0)
6876 end
6877 end
6878 return head
6879 end
6880 end
6881 }}%

```

```

6882 \fi
6883 % == transforms ==
6884 \ifx\bbk@KVP@transforms\@nnil\else
6885 \def\bbk@elt##1##2##3{%
6886 \in@{$transforms.}{##1}%
6887 \ifin@
6888 \def\bbk@tempa{##1}%
6889 \bbk@replace\bbk@tempa{transforms.}{}%
6890 \bbk@carg\bbk@transforms{babel\bbk@tempa}{##2}{##3}%
6891 \fi}%
6892 \bbk@exp{%
6893 \\\bbk@ifblank{\bbk@cl{dgnat}}}%
6894 {\let\\bbk@tempa\relax}%
6895 {\def\\bbk@tempa{%
6896 \\\bbk@elt{transforms.prehyphenation}%
6897 {digits.native.1.0}{([0-9])}%
6898 \\\bbk@elt{transforms.prehyphenation}%
6899 {digits.native.1.1}{string={\string|0123456789\string|\bbk@cl{dgnat}}}}}%
6900 \ifx\bbk@tempa\relax\else
6901 \toks@{\expandafter\expandafter\expandafter{%
6902 \csname bbl@inidata@\language\endcsname}%
6903 \bbk@csarg\edef{inidata@\language}{%
6904 \unexpanded\expandafter{\bbk@tempa}%
6905 \the\toks@}%
6906 \fi
6907 \csname bbl@inidata@\language\endcsname
6908 \bbk@release@transforms\relax % \relax closes the last item.
6909 \fi}

```

Start tabular here:

```

6910 \def\localerestoredirs{%
6911 \ifcase\bbk@thetextdir
6912 \ifnum\textdirection=\z@\else\textdir TLT\fi
6913 \else
6914 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6915 \fi
6916 \ifcase\bbk@thepardir
6917 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6918 \else
6919 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6920 \fi}
6921 %
6922 \IfBabelLayout{tabular}%
6923 {\chardef\bbk@tabular@mode\tw}% All RTL
6924 {\IfBabelLayout{notabular}%
6925 {\chardef\bbk@tabular@mode\z}%
6926 {\chardef\bbk@tabular@mode\@ne}}% Mixed, with LTR cols
6927 %
6928 \ifnum\bbk@bidimode>\@ne % Any lua bidi= except default=1
6929 % Redefine: vrules mess up dirs.
6930 \def\@arstrut{\relax\copy\@arstrutbox}%
6931 \ifcase\bbk@tabular@mode\or % 1 = Mixed - default
6932 \let\bbk@parabefore\relax
6933 \AddToHook{para/before}{\bbk@parabefore}
6934 \AtBeginDocument{%
6935 \bbk@replace\@tabular{$}{%
6936 \def\bbk@insidemath{0}%
6937 \def\bbk@parabefore{\localerestoredirs}}}%
6938 \ifnum\bbk@tabular@mode=\@ne
6939 \bbk@ifunset{tabclassz}{%
6940 \bbk@exp{% Hide conditionals
6941 \\\bbk@sreplace\\@tabclassz
6942 {\<ifcase>\\@chnum}%

```

```

6943         {\localerestoredirs\<ifcase>\\@chnum}}}%
6944 \@ifpackageloaded{colortbl}%
6945     {\bbl@sreplace\@classz
6946     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6947 {\@ifpackageloaded{array}%
6948     {\bbl@exp{% Hide conditionals
6949         \\bbl@sreplace\\@classz
6950         {\<ifcase>\\@chnum}%
6951         {\bgroup\\localerestoredirs\<ifcase>\\@chnum}%
6952         \\bbl@sreplace\\@classz
6953         {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6954     {}}}%
6955 \fi}%
6956 \or % 2 = All RTL - tabular
6957 \let\bbl@parabefore\relax
6958 \AddToHook{para/before}{\bbl@parabefore}%
6959 \AtBeginDocument{%
6960     \@ifpackageloaded{colortbl}%
6961     {\bbl@replace\@tabular{$}{$}%
6962     \def\bbl@insidemath{0}%
6963     \def\bbl@parabefore{\localerestoredirs}}}%
6964     \bbl@sreplace\@classz
6965     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6966     {}}}%
6967 \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.

```

6968 \AtBeginDocument{%
6969     \@ifpackageloaded{multicol}%
6970     {\toks@\expandafter{\multi@column@out}%
6971     \edef\multi@column@out{\bodydir\pagedir\the\toks@}}}%
6972     {}}%
6973 \@ifpackageloaded{paracol}%
6974     {\edef\pcol@output{%
6975     \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6976     {}}}%
6977 \fi

```

Finish here if there in no layout.

```

6978 \ifx\bbl@opt@layout\@nnil\endinput\fi

```

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

```

6979 \ifnum\bbl@bidimode>\z@ % Any bidi=
6980 \def\bbl@nextfake#1% non-local changes, use always inside a group!
6981     \bbl@exp{%
6982         \mathdir\the\bodydir
6983         #1% Once entered in math, set boxes to restore values
6984         \def\\bbl@insidemath{0}%
6985         \<ifmmode>%
6986             \everyvbox{%
6987                 \the\everyvbox
6988                 \bodydir\the\bodydir
6989                 \mathdir\the\mathdir
6990                 \everyhbox{\the\everyhbox}%
6991                 \everyvbox{\the\everyvbox}}}%
6992             \everyhbox{%
6993                 \the\everyhbox
6994                 \bodydir\the\bodydir

```



```

6995         \mathdir\the\mathdir
6996         \everyhbox{\the\everyhbox}%
6997         \everyvbox{\the\everyvbox}}%
6998     \<fi>}}%
6999 \IfBabelLayout{nopars}
7000 {}
7001 {\edef\bblopt@layout{\bblopt@layout.pars.}}%
7002 \IfBabelLayout{pars}
7003 {\def\@hangfrom#1{%
7004     \setbox\@tempboxa\hbox{#{#1}}%
7005     \hangindent\wd\@tempboxa
7006     \ifnum\bblopt@getluadir{page}=\bblopt@getluadir{par}\else
7007         \shapemode\@ne
7008     \fi
7009     \noindent\box\@tempboxa}}
7010 {}
7011 \fi
7012 %
7013 \IfBabelLayout{tabular}
7014 {\let\bblopt@tabular\@tabular
7015  \bblopt@replace\@tabular{$}{\bblopt@nextfake$}%
7016  \let\bblopt@NL\@tabular\@tabular
7017  \AtBeginDocument{%
7018      \ifx\bblopt@NL\@tabular\@tabular\else
7019          \bblopt@exp{\in{\bblopt@nextfake}{\@tabular}}}%
7020      \ifin\@else
7021          \bblopt@replace\@tabular{$}{\bblopt@nextfake$}%
7022      \fi
7023      \let\bblopt@NL\@tabular\@tabular
7024  \fi}}
7025 {}
7026 %
7027 \IfBabelLayout{lists}
7028 {\let\bblopt@list\list
7029  \bblopt@sreplace\list{\parshape}{\bblopt@listparshape}%
7030  \let\bblopt@NL\list\list
7031  \def\bblopt@listparshape#1#2#3{%
7032      \parshape #1 #2 #3 %
7033      \ifnum\bblopt@getluadir{page}=\bblopt@getluadir{par}\else
7034          \shapemode\tw@
7035      \fi}}
7036 {}
7037 %
7038 \IfBabelLayout{graphics}
7039 {\let\bblopt@pictresetdir\relax
7040  \def\bblopt@pictsetdir#1{%
7041      \ifcase\bblopt@thetextdir
7042          \let\bblopt@pictresetdir\relax
7043      \else
7044          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7045              \or\textdir TLT
7046              \else\bodydir TLT \textdir TLT
7047          \fi
7048          % \(\text|par)dir required in pgf:
7049          \def\bblopt@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7050      \fi}%
7051  \AddToHook{env/picture/begin}{\bblopt@pictsetdir\tw@}%
7052  \directlua{
7053      Babel.get_picture_dir = true
7054      Babel.picture_has_bidi = 0
7055      %
7056      function Babel.picture_dir (head)
7057          if not Babel.get_picture_dir then return head end

```

```

7058     if Babel.hlist_has_bidi(head) then
7059         Babel.picture_has_bidi = 1
7060     end
7061     return head
7062 end
7063 \latexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7064 "Babel.picture_dir")
7065 }%
7066 \AtBeginDocument{%
7067     \def\LS@rot{%
7068         \setbox\@outputbox\vbox{%
7069             \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
7070     \long\def\put(#1,#2)#3{%
7071         \@killglue
7072         % Try:
7073         \ifx\bbbl@pictresetdir\relax
7074             \def\bbbl@tempc{0}%
7075         \else
7076             \directlua{
7077                 Babel.get_picture_dir = true
7078                 Babel.picture_has_bidi = 0
7079             }%
7080             \setbox\z@\hb@xt@\z@{%
7081                 \@defaultunitsset\@tempdimc{#1}\unitlength
7082                 \kern\@tempdimc
7083                 #3\hss}%
7084             \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7085         \fi
7086         % Do:
7087         \@defaultunitsset\@tempdimc{#2}\unitlength
7088         \raise\@tempdimc\hb@xt@\z@{%
7089             \@defaultunitsset\@tempdimc{#1}\unitlength
7090             \kern\@tempdimc
7091             {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
7092         \ignorespaces}%
7093     \MakeRobust\put}%
7094 \AtBeginDocument
7095 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
7096 \ifx\pgfpicture\undefined\else
7097     \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
7098     \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
7099     \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
7100 \fi
7101 \ifx\tikzpicture\undefined\else
7102     \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
7103     \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
7104     \bbbl@sreplace\tikz{\beginpgpreamble}\beginpgpreamble\bbbl@pictsetdir\tw@}%
7105     \bbbl@sreplace\tikzpicture{\beginpgpreamble}\beginpgpreamble\bbbl@pictsetdir\tw@}%
7106 \fi
7107 \ifx\tcolorbox\undefined\else
7108     \def\tcb@drawing@env@begin{%
7109         \csname tcb@before@tcb@split@state\endcsname
7110         \bbbl@pictsetdir\tw@
7111         \begin{kvtcb@graphenv}%
7112         \tcb@bbdraw
7113         \tcb@apply@graph@patches}%
7114     \def\tcb@drawing@env@end{%
7115         \end{kvtcb@graphenv}%
7116         \bbbl@pictresetdir
7117         \csname tcb@after@tcb@split@state\endcsname}%
7118     \fi
7119 }}
7120 {}

```

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`.

```

7121 \IfBabelLayout{counters*}%
7122   {\bbl@add\bbl@opt@layout{.counters.}}%
7123   \directlua{
7124     luatexbase.add_to_callback("process_output_buffer",
7125       Babel.discard_sublr , "Babel.discard_sublr") }%
7126   }{}
7127 \IfBabelLayout{counters}%
7128   {\let\bbl@0L@@textsuperscript\@textsuperscript
7129     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7130     \let\bbl@latinarabic=\@arabic
7131     \let\bbl@0L@@arabic\@arabic
7132     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7133     \ifpackagewith{babel}{bidi=default}%
7134       {\let\bbl@asciroman=\@roman
7135         \let\bbl@0L@@roman\@roman
7136         \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7137         \let\bbl@asciiRoman=\@Roman
7138         \let\bbl@0L@@roman\@Roman
7139         \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7140         \let\bbl@0L@labelenumii\labelenumii
7141         \def\labelenumii{}\theenumii}%
7142         \let\bbl@0L@p@enumiii\p@enumiii
7143         \def\p@enumiii{\p@enumii}\theenumii{}}{}{}{}

```

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

```

7144 \IfBabelLayout{extras}%
7145   {\bbl@ncarg\let\bbl@0L@underline{underline }%
7146     \bbl@carg\bbl@sreplace{underline }%
7147     {$\@@underline}{\bgroup\bbl@nextfake$\@@underline}%
7148     \bbl@carg\bbl@sreplace{underline }%
7149     {\m@th$}{\m@th$\egroup}%
7150     \let\bbl@0L@LaTeXe\LaTeXe
7151     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7152       \if b\expandafter\@car\@series\@nil\boldmath\fi
7153       \babelsublr{%
7154         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
7155   }{}
7156 \end{luatex}

```

10.13 Lua: transforms

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

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

```

7157 (*transforms)
7158 Babel.linebreaking.replacements = {}
7159 Babel.linebreaking.replacements[0] = {} -- pre
7160 Babel.linebreaking.replacements[1] = {} -- post
7161
7162 function Babel.toval(v)

```

```

7163 if type(v) == 'table' then
7164     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7165 else
7166     return v
7167 end
7168 end
7169
7170 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7171
7172 function Babel.set_hboxed(head, gc)
7173     for item in node.traverse(head) do
7174         node.set_attribute(item, Babel.attr_hboxed, 1)
7175     end
7176     return head
7177 end
7178
7179 Babel.fetch_subtext = {}
7180
7181 Babel.ignore_pre_char = function(node)
7182     return (node.lang == Babel.nohyphenation)
7183 end
7184
7185 Babel.show_transforms = false
7186
7187 -- Merging both functions doesn't seem feasible, because there are too
7188 -- many differences.
7189 Babel.fetch_subtext[0] = function(head)
7190     local word_string = ''
7191     local word_nodes = {}
7192     local lang
7193     local item = head
7194     local inmath = false
7195
7196     while item do
7197
7198         if item.id == 11 then
7199             inmath = (item.subtype == 0)
7200         end
7201
7202         if inmath then
7203             -- pass
7204
7205         elseif item.id == 29 then
7206             local locale = node.get_attribute(item, Babel.attr_locale)
7207
7208             if lang == locale or lang == nil then
7209                 lang = lang or locale
7210                 if Babel.ignore_pre_char(item) then
7211                     word_string = word_string .. Babel.us_char
7212                 else
7213                     if node.has_attribute(item, Babel.attr_hboxed) then
7214                         word_string = word_string .. Babel.us_char
7215                     else
7216                         word_string = word_string .. unicode.utf8.char(item.char)
7217                     end
7218                 end
7219                 word_nodes[#word_nodes+1] = item
7220             else
7221                 break
7222             end
7223
7224         elseif item.id == 12 and item.subtype == 13 then
7225             if node.has_attribute(item, Babel.attr_hboxed) then

```

```

7226     word_string = word_string .. Babel.us_char
7227 else
7228     word_string = word_string .. ' '
7229 end
7230 word_nodes[#word_nodes+1] = item
7231
7232 -- Ignore leading unrecognized nodes, too.
7233 elseif word_string ~= '' then
7234     word_string = word_string .. Babel.us_char
7235     word_nodes[#word_nodes+1] = item -- Will be ignored
7236 end
7237
7238 item = item.next
7239 end
7240
7241 -- Here and above we remove some trailing chars but not the
7242 -- corresponding nodes. But they aren't accessed.
7243 if word_string:sub(-1) == ' ' then
7244     word_string = word_string:sub(1,-2)
7245 end
7246 if Babel.show_transforms then texio.write_nl(word_string) end
7247 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7248 return word_string, word_nodes, item, lang
7249 end
7250
7251 Babel.fetch_subtext[1] = function(head)
7252     local word_string = ''
7253     local word_nodes = {}
7254     local lang
7255     local item = head
7256     local inmath = false
7257
7258     while item do
7259
7260         if item.id == 11 then
7261             inmath = (item.subtype == 0)
7262         end
7263
7264         if inmath then
7265             -- pass
7266         end
7267
7268         elseif item.id == 29 then
7269             if item.lang == lang or lang == nil then
7270                 lang = lang or item.lang
7271                 if node.has_attribute(item, Babel.attr_hboxed) then
7272                     word_string = word_string .. Babel.us_char
7273                     elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7274                         word_string = word_string .. Babel.us_char
7275                     else
7276                         word_string = word_string .. unicode.utf8.char(item.char)
7277                     end
7278                     word_nodes[#word_nodes+1] = item
7279                 else
7280                     break
7281                 end
7282
7283             elseif item.id == 7 and item.subtype == 2 then
7284                 if node.has_attribute(item, Babel.attr_hboxed) then
7285                     word_string = word_string .. Babel.us_char
7286                 else
7287                     word_string = word_string .. '='
7288                 end
7289                 word_nodes[#word_nodes+1] = item

```

```

7289
7290     elseif item.id == 7 and item.subtype == 3 then
7291         if node.has_attribute(item, Babel.attr_hboxed) then
7292             word_string = word_string .. Babel.us_char
7293         else
7294             word_string = word_string .. '|'
7295         end
7296         word_nodes[#word_nodes+1] = item
7297
7298         -- (1) Go to next word if nothing was found, and (2) implicitly
7299         -- remove leading USs.
7300     elseif word_string == '' then
7301         -- pass
7302
7303         -- This is the responsible for splitting by words.
7304     elseif (item.id == 12 and item.subtype == 13) then
7305         break
7306
7307     else
7308         word_string = word_string .. Babel.us_char
7309         word_nodes[#word_nodes+1] = item -- Will be ignored
7310     end
7311
7312     item = item.next
7313 end
7314 if Babel.show_transforms then texio.write_nl(word_string) end
7315 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7316 return word_string, word_nodes, item, lang
7317 end
7318
7319 function Babel.pre_hyphenate_replace(head)
7320     Babel.hyphenate_replace(head, 0)
7321 end
7322
7323 function Babel.post_hyphenate_replace(head)
7324     Babel.hyphenate_replace(head, 1)
7325 end
7326
7327 Babel.us_char = string.char(31)
7328
7329 function Babel.hyphenate_replace(head, mode)
7330     local u = unicode.utf8
7331     local lbkr = Babel.linebreaking.replacements[mode]
7332     local tovalue = Babel.tovalue
7333
7334     local word_head = head
7335
7336     if Babel.show_transforms then
7337         texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7338     end
7339
7340     while true do -- for each subtext block
7341
7342         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7343
7344         if Babel.debug then
7345             print()
7346             print((mode == 0) and '@@@<' or '@@@>', w)
7347         end
7348
7349         if nw == nil and w == '' then break end
7350
7351         if not lang then goto next end

```

```

7352 if not lbkr[lang] then goto next end
7353
7354 -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7355 -- loops are nested.
7356 for k=1, #lbkr[lang] do
7357     local p = lbkr[lang][k].pattern
7358     local r = lbkr[lang][k].replace
7359     local attr = lbkr[lang][k].attr or -1
7360
7361     if Babel.debug then
7362         print('*****', p, mode)
7363     end
7364
7365     -- This variable is set in some cases below to the first *byte*
7366     -- after the match, either as found by u.match (faster) or the
7367     -- computed position based on sc if w has changed.
7368     local last_match = 0
7369     local step = 0
7370
7371     -- For every match.
7372     while true do
7373         if Babel.debug then
7374             print('====')
7375         end
7376         local new -- used when inserting and removing nodes
7377         local dummy_node -- used by after
7378
7379         local matches = { u.match(w, p, last_match) }
7380
7381         if #matches < 2 then break end
7382
7383         -- Get and remove empty captures (with ()'s, which return a
7384         -- number with the position), and keep actual captures
7385         -- (from (...)), if any, in matches.
7386         local first = table.remove(matches, 1)
7387         local last = table.remove(matches, #matches)
7388         -- Non re-fetched substrings may contain \31, which separates
7389         -- subsubstrings.
7390         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7391
7392         local save_last = last -- with A()BC()D, points to D
7393
7394         -- Fix offsets, from bytes to unicode. Explained above.
7395         first = u.len(w:sub(1, first-1)) + 1
7396         last = u.len(w:sub(1, last-1)) -- now last points to C
7397
7398         -- This loop stores in a small table the nodes
7399         -- corresponding to the pattern. Used by 'data' to provide a
7400         -- predictable behavior with 'insert' (w_nodes is modified on
7401         -- the fly), and also access to 'remove'd nodes.
7402         local sc = first-1 -- Used below, too
7403         local data_nodes = {}
7404
7405         local enabled = true
7406         for q = 1, last-first+1 do
7407             data_nodes[q] = w_nodes[sc+q]
7408             if enabled
7409                 and attr > -1
7410                 and not node.has_attribute(data_nodes[q], attr)
7411             then
7412                 enabled = false
7413             end
7414         end

```

```

7415
7416 -- This loop traverses the matched substring and takes the
7417 -- corresponding action stored in the replacement list.
7418 -- sc = the position in substr nodes / string
7419 -- rc = the replacement table index
7420 local rc = 0
7421
7422 ----- TODO. dummy_node?
7423 while rc < last-first+1 or dummy_node do -- for each replacement
7424     if Babel.debug then
7425         print('.....', rc + 1)
7426     end
7427     sc = sc + 1
7428     rc = rc + 1
7429
7430     if Babel.debug then
7431         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7432         local ss = ''
7433         for itt in node.traverse(head) do
7434             if itt.id == 29 then
7435                 ss = ss .. unicode.utf8.char(itt.char)
7436             else
7437                 ss = ss .. '{' .. itt.id .. '}'
7438             end
7439         end
7440         print('*****', ss)
7441
7442     end
7443
7444     local crep = r[rc]
7445     local item = w_nodes[sc]
7446     local item_base = item
7447     local placeholder = Babel.us_char
7448     local d
7449
7450     if crep and crep.data then
7451         item_base = data_nodes[crep.data]
7452     end
7453
7454     if crep then
7455         step = crep.step or step
7456     end
7457
7458     if crep and crep.after then
7459         crep.insert = true
7460         if dummy_node then
7461             item = dummy_node
7462         else -- TODO. if there is a node after?
7463             d = node.copy(item_base)
7464             head, item = node.insert_after(head, item, d)
7465             dummy_node = item
7466         end
7467     end
7468
7469     if crep and not crep.after and dummy_node then
7470         node.remove(head, dummy_node)
7471         dummy_node = nil
7472     end
7473
7474     if not enabled then
7475         last_match = save_last
7476         goto next
7477

```



```

7478     elseif crep and next(crep) == nil then -- = {}
7479         if step == 0 then
7480             last_match = save_last    -- Optimization
7481         else
7482             last_match = utf8.offset(w, sc+step)
7483         end
7484         goto next
7485
7486     elseif crep == nil or crep.remove then
7487         node.remove(head, item)
7488         table.remove(w_nodes, sc)
7489         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7490         sc = sc - 1 -- Nothing has been inserted.
7491         last_match = utf8.offset(w, sc+1+step)
7492         goto next
7493
7494     elseif crep and crep.kashida then -- Experimental
7495         node.set_attribute(item,
7496             Babel.attr_kashida,
7497             crep.kashida)
7498         last_match = utf8.offset(w, sc+1+step)
7499         goto next
7500
7501     elseif crep and crep.string then
7502         local str = crep.string(matches)
7503         if str == '' then -- Gather with nil
7504             node.remove(head, item)
7505             table.remove(w_nodes, sc)
7506             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7507             sc = sc - 1 -- Nothing has been inserted.
7508         else
7509             local loop_first = true
7510             for s in string.utfvalues(str) do
7511                 d = node.copy(item_base)
7512                 d.char = s
7513                 if loop_first then
7514                     loop_first = false
7515                     head, new = node.insert_before(head, item, d)
7516                     if sc == 1 then
7517                         word_head = head
7518                     end
7519                     w_nodes[sc] = d
7520                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7521                 else
7522                     sc = sc + 1
7523                     head, new = node.insert_before(head, item, d)
7524                     table.insert(w_nodes, sc, new)
7525                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7526                 end
7527                 if Babel.debug then
7528                     print('.....', 'str')
7529                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7530                 end
7531             end -- for
7532             node.remove(head, item)
7533         end -- if ''
7534         last_match = utf8.offset(w, sc+1+step)
7535         goto next
7536
7537     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7538         d = node.new(7, 3) -- (disc, regular)
7539         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7540         d.post = Babel.str_to_nodes(crep.post, matches, item_base)

```

```

7541     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7542     d.attr = item_base.attr
7543     if crep.pre == nil then -- TeXbook p96
7544         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7545     else
7546         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7547     end
7548     placeholder = '|'
7549     head, new = node.insert_before(head, item, d)
7550
7551 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7552     -- ERROR
7553
7554 elseif crep and crep.penalty then
7555     d = node.new(14, 0) -- (penalty, userpenalty)
7556     d.attr = item_base.attr
7557     d.penalty = tovalue(crep.penalty)
7558     head, new = node.insert_before(head, item, d)
7559
7560 elseif crep and crep.space then
7561     -- 655360 = 10 pt = 10 * 65536 sp
7562     d = node.new(12, 13) -- (glue, spaceskip)
7563     local quad = font.getfont(item_base.font).size or 655360
7564     node.setglue(d, tovalue(crep.space[1]) * quad,
7565                   tovalue(crep.space[2]) * quad,
7566                   tovalue(crep.space[3]) * quad)
7567     if mode == 0 then
7568         placeholder = ' '
7569     end
7570     head, new = node.insert_before(head, item, d)
7571
7572 elseif crep and crep.norule then
7573     -- 655360 = 10 pt = 10 * 65536 sp
7574     d = node.new(2, 3) -- (rule, empty) = \no*rule
7575     local quad = font.getfont(item_base.font).size or 655360
7576     d.width = tovalue(crep.norule[1]) * quad
7577     d.height = tovalue(crep.norule[2]) * quad
7578     d.depth = tovalue(crep.norule[3]) * quad
7579     head, new = node.insert_before(head, item, d)
7580
7581 elseif crep and crep.spacefactor then
7582     d = node.new(12, 13) -- (glue, spaceskip)
7583     local base_font = font.getfont(item_base.font)
7584     node.setglue(d,
7585                 tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7586                 tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7587                 tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7588     if mode == 0 then
7589         placeholder = ' '
7590     end
7591     head, new = node.insert_before(head, item, d)
7592
7593 elseif mode == 0 and crep and crep.space then
7594     -- ERROR
7595
7596 elseif crep and crep.kern then
7597     d = node.new(13, 1) -- (kern, user)
7598     local quad = font.getfont(item_base.font).size or 655360
7599     d.attr = item_base.attr
7600     d.kern = tovalue(crep.kern) * quad
7601     head, new = node.insert_before(head, item, d)
7602
7603 elseif crep and crep.node then

```

```

7604         d = node.new(crep.node[1], crep.node[2])
7605         d.attr = item_base.attr
7606         head, new = node.insert_before(head, item, d)
7607
7608     end -- i.e., replacement cases
7609
7610     -- Shared by disc, space(factor), kern, node and penalty.
7611     if sc == 1 then
7612         word_head = head
7613     end
7614     if crep.insert then
7615         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7616         table.insert(w_nodes, sc, new)
7617         last = last + 1
7618     else
7619         w_nodes[sc] = d
7620         node.remove(head, item)
7621         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7622     end
7623
7624     last_match = utf8.offset(w, sc+1+step)
7625
7626     ::next::
7627
7628 end -- for each replacement
7629
7630 if Babel.show_transforms then texio.write_nl('> ' .. w) end
7631 if Babel.debug then
7632     print('.....', '/')
7633     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7634 end
7635
7636 if dummy_node then
7637     node.remove(head, dummy_node)
7638     dummy_node = nil
7639 end
7640
7641 end -- for match
7642
7643 end -- for patterns
7644
7645 ::next::
7646 word_head = nw
7647 end -- for substring
7648
7649 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7650 return head
7651 end
7652
7653 -- This table stores capture maps, numbered consecutively
7654 Babel.capture_maps = {}
7655
7656 -- The following functions belong to the next macro
7657 function Babel.capture_func(key, cap)
7658     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
7659     local cnt
7660     local u = unicode.utf8
7661     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7662     if cnt == 0 then
7663         ret = u.gsub(ret, '{(%x%x%x%x+)}',
7664             function (n)
7665                 return u.char(tonumber(n, 16))
7666             end)

```

```

7667 end
7668 ret = ret:gsub("%[%[%]%].%", '')
7669 ret = ret:gsub("%.%.%[%[%]"]", '')
7670 return key .. [[=function(m) return ]] .. ret .. [[ end]]
7671 end
7672
7673 function Babel.capt_map(from, mapno)
7674 return Babel.capture_maps[mapno][from] or from
7675 end
7676
7677 -- Handle the {n|abc|ABC} syntax in captures
7678 function Babel.capture_func_map(capno, from, to)
7679 local u = unicode.utf8
7680 from = u.gsub(from, '{(%x%x%x%x+)}',
7681 function (n)
7682 return u.char(tonumber(n, 16))
7683 end)
7684 to = u.gsub(to, '{(%x%x%x%x+)}',
7685 function (n)
7686 return u.char(tonumber(n, 16))
7687 end)
7688 local froms = {}
7689 for s in string.utfcharacters(from) do
7690 table.insert(froms, s)
7691 end
7692 local cnt = 1
7693 table.insert(Babel.capture_maps, {})
7694 local mlen = table.getn(Babel.capture_maps)
7695 for s in string.utfcharacters(to) do
7696 Babel.capture_maps[mlen][froms[cnt]] = s
7697 cnt = cnt + 1
7698 end
7699 return "]"..Babel.capt_map(m[" .. capno .. "], " ..
7700 (mlen) .. ").." .. "["
7701 end
7702
7703 -- Create/Extend reversed sorted list of kashida weights:
7704 function Babel.capture_kashida(key, wt)
7705 wt = tonumber(wt)
7706 if Babel.kashida_wts then
7707 for p, q in ipairs(Babel.kashida_wts) do
7708 if wt == q then
7709 break
7710 elseif wt > q then
7711 table.insert(Babel.kashida_wts, p, wt)
7712 break
7713 elseif table.getn(Babel.kashida_wts) == p then
7714 table.insert(Babel.kashida_wts, wt)
7715 end
7716 end
7717 else
7718 Babel.kashida_wts = { wt }
7719 end
7720 return 'kashida = ' .. wt
7721 end
7722
7723 function Babel.capture_node(id, subtype)
7724 local sbt = 0
7725 for k, v in pairs(node.subtypes(id)) do
7726 if v == subtype then sbt = k end
7727 end
7728 return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7729 end

```

```

7730
7731 -- Experimental: applies prehyphenation transforms to a string (letters
7732 -- and spaces).
7733 function Babel.string_prehyphenation(str, locale)
7734   local n, head, last, res
7735   head = node.new(8, 0) -- dummy (hack just to start)
7736   last = head
7737   for s in string.utfvalues(str) do
7738     if s == 20 then
7739       n = node.new(12, 0)
7740     else
7741       n = node.new(29, 0)
7742       n.char = s
7743     end
7744     node.set_attribute(n, Babel.attr_locale, locale)
7745     last.next = n
7746     last = n
7747   end
7748   head = Babel.hyphenate_replace(head, 0)
7749   res = ''
7750   for n in node.traverse(head) do
7751     if n.id == 12 then
7752       res = res .. ' '
7753     elseif n.id == 29 then
7754       res = res .. unicode.utf8.char(n.char)
7755     end
7756   end
7757   tex.print(res)
7758 end
7759  $\langle$ transforms $\rangle$ 

```

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.

```

7760 (*basic-r)
7761 Babel.bidi_enabled = true
7762
7763 require('babel-data-bidi.lua')
7764
7765 local characters = Babel.characters
7766 local ranges = Babel.ranges
7767
7768 local DIR = node.id("dir")
7769
7770 local function dir_mark(head, from, to, outer)
7771   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7772   local d = node.new(DIR)
7773   d.dir = '+' .. dir
7774   node.insert_before(head, from, d)
7775   d = node.new(DIR)
7776   d.dir = '-' .. dir
7777   node.insert_after(head, to, d)
7778 end
7779
7780 function Babel.bidi(head, ispar)
7781   local first_n, last_n          -- first and last char with nums
7782   local last_es                 -- an auxiliary 'last' used with nums
7783   local first_d, last_d         -- first and last char in L/R block
7784   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):

```

7785   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7786   local strong_lr = (strong == 'l') and 'l' or 'r'
7787   local outer = strong
7788
7789   local new_dir = false
7790   local first_dir = false
7791   local inmath = false
7792
7793   local last_lr
7794
7795   local type_n = ''
7796
7797   for item in node.traverse(head) do
7798
7799     -- three cases: glyph, dir, otherwise
7800     if item.id == node.id'glyph'
7801       or (item.id == 7 and item.subtype == 2) then
7802
7803       local itemchar
7804       if item.id == 7 and item.subtype == 2 then
7805         itemchar = item.replace.char
7806       else
7807         itemchar = item.char
7808       end
7809       local chardata = characters[itemchar]
7810       dir = chardata and chardata.d or nil
7811       if not dir then

```

```

7812         for nn, et in ipairs(ranges) do
7813             if itemchar < et[1] then
7814                 break
7815             elseif itemchar <= et[2] then
7816                 dir = et[3]
7817                 break
7818             end
7819         end
7820     end
7821     dir = dir or 'l'
7822     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).

```

7823     if new_dir then
7824         attr_dir = 0
7825         for at in node.traverse(item.attr) do
7826             if at.number == Babel.attr_dir then
7827                 attr_dir = at.value & 0x3
7828             end
7829         end
7830         if attr_dir == 1 then
7831             strong = 'r'
7832         elseif attr_dir == 2 then
7833             strong = 'al'
7834         else
7835             strong = 'l'
7836         end
7837         strong_lr = (strong == 'l') and 'l' or 'r'
7838         outer = strong_lr
7839         new_dir = false
7840     end
7841
7842     if dir == 'nsm' then dir = strong end -- W1

```

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

```

7843     dir_real = dir -- We need dir_real to set strong below
7844     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:

```

7845     if strong == 'al' then
7846         if dir == 'en' then dir = 'an' end -- W2
7847         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7848         strong_lr = 'r' -- W3
7849     end

```

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

```

7850     elseif item.id == node.id'dir' and not inmath then
7851         new_dir = true
7852         dir = nil
7853     elseif item.id == node.id'math' then
7854         inmath = (item.subtype == 0)
7855     else
7856         dir = nil -- Not a char
7857     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>.

```

7858   if dir == 'en' or dir == 'an' or dir == 'et' then
7859       if dir ~= 'et' then
7860           type_n = dir
7861       end
7862       first_n = first_n or item
7863       last_n = last_es or item
7864       last_es = nil
7865   elseif dir == 'es' and last_n then -- W3+W6
7866       last_es = item
7867   elseif dir == 'cs' then -- it's right - do nothing
7868   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7869       if strong_lr == 'r' and type_n ~= '' then
7870           dir_mark(head, first_n, last_n, 'r')
7871       elseif strong_lr == 'l' and first_d and type_n == 'an' then
7872           dir_mark(head, first_n, last_n, 'r')
7873           dir_mark(head, first_d, last_d, outer)
7874           first_d, last_d = nil, nil
7875       elseif strong_lr == 'l' and type_n ~= '' then
7876           last_d = last_n
7877       end
7878       type_n = ''
7879       first_n, last_n = nil, nil
7880   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:

```

7881   if dir == 'l' or dir == 'r' then
7882       if dir ~= outer then
7883           first_d = first_d or item
7884           last_d = item
7885       elseif first_d and dir ~= strong_lr then
7886           dir_mark(head, first_d, last_d, outer)
7887           first_d, last_d = nil, nil
7888       end
7889   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.

```

7890   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7891       item.char = characters[item.char] and
7892           characters[item.char].m or item.char
7893   elseif (dir or new_dir) and last_lr ~= item then
7894       local mir = outer .. strong_lr .. (dir or outer)
7895       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7896           for ch in node.traverse(node.next(last_lr)) do
7897               if ch == item then break end
7898               if ch.id == node.id'glyph' and characters[ch.char] then
7899                   ch.char = characters[ch.char].m or ch.char
7900               end
7901           end
7902       end
7903   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).

```

7904   if dir == 'l' or dir == 'r' then

```



```

7905     last_lr = item
7906     strong = dir_real          -- Don't search back - best save now
7907     strong_lr = (strong == 'l') and 'l' or 'r'
7908     elseif new_dir then
7909         last_lr = nil
7910     end
7911 end

```

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

```

7912 if last_lr and outer == 'r' then
7913     for ch in node.traverse_id(node.id('glyph', node.next(last_lr))) do
7914         if characters[ch.char] then
7915             ch.char = characters[ch.char].m or ch.char
7916         end
7917     end
7918 end
7919 if first_n then
7920     dir_mark(head, first_n, last_n, outer)
7921 end
7922 if first_d then
7923     dir_mark(head, first_d, last_d, outer)
7924 end

```

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

```

7925 return node.prev(head) or head
7926 end
7927 </basic-r>

```

And here the Lua code for bidi=basic:

```

7928 (*basic)
7929 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7930
7931 Babel.fontmap = Babel.fontmap or {}
7932 Babel.fontmap[0] = {}          -- l
7933 Babel.fontmap[1] = {}          -- r
7934 Babel.fontmap[2] = {}          -- al/an
7935
7936 -- To cancel mirroring. Also OML, OMS, U?
7937 Babel.symbol_fonts = Babel.symbol_fonts or {}
7938 Babel.symbol_fonts[font.id('tenln')] = true
7939 Babel.symbol_fonts[font.id('tenlnw')] = true
7940 Babel.symbol_fonts[font.id('tencirc')] = true
7941 Babel.symbol_fonts[font.id('tencircw')] = true
7942
7943 Babel.bidi_enabled = true
7944 Babel.mirroring_enabled = true
7945
7946 require('babel-data-bidi.lua')
7947
7948 local characters = Babel.characters
7949 local ranges = Babel.ranges
7950
7951 local DIR = node.id('dir')
7952 local GLYPH = node.id('glyph')
7953
7954 local function insert_implicit(head, state, outer)
7955     local new_state = state
7956     if state.sim and state.eim and state.sim ~= state.eim then
7957         dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7958         local d = node.new(DIR)
7959         d.dir = '+' .. dir
7960         node.insert_before(head, state.sim, d)
7961         local d = node.new(DIR)

```

```

7962     d.dir = '-' .. dir
7963     node.insert_after(head, state.eim, d)
7964 end
7965 new_state.sim, new_state.eim = nil, nil
7966 return head, new_state
7967 end
7968
7969 local function insert_numeric(head, state)
7970     local new
7971     local new_state = state
7972     if state.san and state.ean and state.san ~= state.ean then
7973         local d = node.new(DIR)
7974         d.dir = '+TLT'
7975         _, new = node.insert_before(head, state.san, d)
7976         if state.san == state.sim then state.sim = new end
7977         local d = node.new(DIR)
7978         d.dir = '-TLT'
7979         _, new = node.insert_after(head, state.ean, d)
7980         if state.ean == state.eim then state.eim = new end
7981     end
7982     new_state.san, new_state.ean = nil, nil
7983     return head, new_state
7984 end
7985
7986 local function glyph_not_symbol_font(node)
7987     if node.id == GLYPH then
7988         return not Babel.symbol_fonts[node.font]
7989     else
7990         return false
7991     end
7992 end
7993
7994 -- TODO - \hbox with an explicit dir can lead to wrong results
7995 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7996 -- was made to improve the situation, but the problem is the 3-dir
7997 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7998 -- well.
7999
8000 function Babel.bidi(head, ispar, hdir)
8001     local d -- d is used mainly for computations in a loop
8002     local prev_d = ''
8003     local new_d = false
8004
8005     local nodes = {}
8006     local outer_first = nil
8007     local inmath = false
8008
8009     local glue_d = nil
8010     local glue_i = nil
8011
8012     local has_en = false
8013     local first_et = nil
8014
8015     local has_hyperlink = false
8016
8017     local ATDIR = Babel.attr_dir
8018     local attr_d, temp
8019     local locale_d
8020
8021     local save_outer
8022     local locale_d = node.get_attribute(head, ATDIR)
8023     if locale_d then
8024         locale_d = locale_d & 0x3

```

```

8025     save_outer = (locale_d == 0 and 'l') or
8026                   (locale_d == 1 and 'r') or
8027                   (locale_d == 2 and 'al')
8028 elseif ispar then      -- Or error? Shouldn't happen
8029     -- when the callback is called, we are just _after_ the box,
8030     -- and the textdir is that of the surrounding text
8031     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8032 else                   -- Empty box
8033     save_outer = ('TRT' == hdir) and 'r' or 'l'
8034 end
8035 local outer = save_outer
8036 local last = outer
8037 -- 'al' is only taken into account in the first, current loop
8038 if save_outer == 'al' then save_outer = 'r' end
8039
8040 local fontmap = Babel.fontmap
8041
8042 for item in node.traverse(head) do
8043
8044     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8045     locale_d = node.get_attribute(item, ATDIR)
8046     node.set_attribute(item, ATDIR, 0x80)
8047
8048     -- In what follows, #node is the last (previous) node, because the
8049     -- current one is not added until we start processing the neutrals.
8050     -- three cases: glyph, dir, otherwise
8051     if glyph_not_symbol_font(item)
8052         or (item.id == 7 and item.subtype == 2) then
8053
8054         if locale_d == 0x80 then goto nextnode end
8055
8056         local d_font = nil
8057         local item_r
8058         if item.id == 7 and item.subtype == 2 then
8059             item_r = item.replace      -- automatic discs have just 1 glyph
8060         else
8061             item_r = item
8062         end
8063
8064         local chardata = characters[item_r.char]
8065         d = chardata and chardata.d or nil
8066         if not d or d == 'nsm' then
8067             for nn, et in ipairs(ranges) do
8068                 if item_r.char < et[1] then
8069                     break
8070                 elseif item_r.char <= et[2] then
8071                     if not d then d = et[3]
8072                     elseif d == 'nsm' then d_font = et[3]
8073                     end
8074                     break
8075                 end
8076             end
8077         end
8078         d = d or 'l'
8079
8080         -- A short 'pause' in bidi for mapfont
8081         -- %%% TODO. move if fontmap here
8082         d_font = d_font or d
8083         d_font = (d_font == 'l' and 0) or
8084                 (d_font == 'nsm' and 0) or
8085                 (d_font == 'r' and 1) or
8086                 (d_font == 'al' and 2) or
8087                 (d_font == 'an' and 2) or nil

```

```

8088     if d_font and fontmap and fontmap[d_font][item_r.font] then
8089         item_r.font = fontmap[d_font][item_r.font]
8090     end
8091
8092     if new_d then
8093         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8094         if inmath then
8095             attr_d = 0
8096         else
8097             attr_d = locale_d & 0x3
8098         end
8099         if attr_d == 1 then
8100             outer_first = 'r'
8101             last = 'r'
8102         elseif attr_d == 2 then
8103             outer_first = 'r'
8104             last = 'al'
8105         else
8106             outer_first = 'l'
8107             last = 'l'
8108         end
8109         outer = last
8110         has_en = false
8111         first_et = nil
8112         new_d = false
8113     end
8114
8115     if glue_d then
8116         if (d == 'l' and 'l' or 'r') ~= glue_d then
8117             table.insert(nodes, {glue_i, 'on', nil})
8118         end
8119         glue_d = nil
8120         glue_i = nil
8121     end
8122
8123     elseif item.id == DIR then
8124         d = nil
8125         new_d = true
8126
8127     elseif item.id == node.id'glue' and item.subtype == 13 then
8128         glue_d = d
8129         glue_i = item
8130         d = nil
8131
8132     elseif item.id == node.id'math' then
8133         inmath = (item.subtype == 0)
8134
8135     elseif item.id == 8 and item.subtype == 19 then
8136         has_hyperlink = true
8137
8138     else
8139         d = nil
8140     end
8141
8142     -- AL <= EN/ET/ES      -- W2 + W3 + W6
8143     if last == 'al' and d == 'en' then
8144         d = 'an'          -- W3
8145     elseif last == 'al' and (d == 'et' or d == 'es') then
8146         d = 'on'          -- W6
8147     end
8148
8149     -- EN + CS/ES + EN      -- W4
8150     if d == 'en' and #nodes >= 2 then

```

```

8151     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8152         and nodes[#nodes-1][2] == 'en' then
8153         nodes[#nodes][2] = 'en'
8154     end
8155 end
8156
8157 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8158 if d == 'an' and #nodes >= 2 then
8159     if (nodes[#nodes][2] == 'cs')
8160         and nodes[#nodes-1][2] == 'an' then
8161         nodes[#nodes][2] = 'an'
8162     end
8163 end
8164
8165 -- ET/EN                -- W5 + W7->l / W6->on
8166 if d == 'et' then
8167     first_et = first_et or (#nodes + 1)
8168 elseif d == 'en' then
8169     has_en = true
8170     first_et = first_et or (#nodes + 1)
8171 elseif first_et then    -- d may be nil here !
8172     if has_en then
8173         if last == 'l' then
8174             temp = 'l'    -- W7
8175         else
8176             temp = 'en'   -- W5
8177         end
8178     else
8179         temp = 'on'      -- W6
8180     end
8181     for e = first_et, #nodes do
8182         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8183     end
8184     first_et = nil
8185     has_en = false
8186 end
8187
8188 -- Force mathdir in math if ON (currently works as expected only
8189 -- with 'l')
8190
8191 if inmath and d == 'on' then
8192     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8193 end
8194
8195 if d then
8196     if d == 'al' then
8197         d = 'r'
8198         last = 'al'
8199     elseif d == 'l' or d == 'r' then
8200         last = d
8201     end
8202     prev_d = d
8203     table.insert(nodes, {item, d, outer_first})
8204 end
8205
8206 outer_first = nil
8207
8208 ::nextnode::
8209
8210 end -- for each node
8211
8212 -- TODO -- repeated here in case EN/ET is the last node. Find a
8213 -- better way of doing things:

```

```

8214 if first_et then          -- dir may be nil here !
8215   if has_en then
8216     if last == 'l' then
8217       temp = 'l'    -- W7
8218     else
8219       temp = 'en'    -- W5
8220     end
8221   else
8222     temp = 'on'      -- W6
8223   end
8224   for e = first_et, #nodes do
8225     if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8226   end
8227 end
8228
8229 -- dummy node, to close things
8230 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8231
8232 ----- NEUTRAL -----
8233
8234 outer = save_outer
8235 last = outer
8236
8237 local first_on = nil
8238
8239 for q = 1, #nodes do
8240   local item
8241
8242   local outer_first = nodes[q][3]
8243   outer = outer_first or outer
8244   last = outer_first or last
8245
8246   local d = nodes[q][2]
8247   if d == 'an' or d == 'en' then d = 'r' end
8248   if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8249
8250   if d == 'on' then
8251     first_on = first_on or q
8252   elseif first_on then
8253     if last == d then
8254       temp = d
8255     else
8256       temp = outer
8257     end
8258     for r = first_on, q - 1 do
8259       nodes[r][2] = temp
8260       item = nodes[r][1]    -- MIRRORING
8261       if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8262         and temp == 'r' and characters[item.char] then
8263         local font_mode = ''
8264         if item.font > 0 and font.fonts[item.font].properties then
8265           font_mode = font.fonts[item.font].properties.mode
8266         end
8267         if font_mode ~= 'harf' and font_mode ~= 'plug' then
8268           item.char = characters[item.char].m or item.char
8269         end
8270       end
8271     end
8272     first_on = nil
8273   end
8274
8275   if d == 'r' or d == 'l' then last = d end
8276 end

```

```

8277
8278 ----- IMPLICIT, REORDER -----
8279
8280 outer = save_outer
8281 last = outer
8282
8283 local state = {}
8284 state.has_r = false
8285
8286 for q = 1, #nodes do
8287
8288     local item = nodes[q][1]
8289
8290     outer = nodes[q][3] or outer
8291
8292     local d = nodes[q][2]
8293
8294     if d == 'nsm' then d = last end          -- W1
8295     if d == 'en' then d = 'an' end
8296     local isdir = (d == 'r' or d == 'l')
8297
8298     if outer == 'l' and d == 'an' then
8299         state.san = state.san or item
8300         state.ean = item
8301     elseif state.san then
8302         head, state = insert_numeric(head, state)
8303     end
8304
8305     if outer == 'l' then
8306         if d == 'an' or d == 'r' then      -- im -> implicit
8307             if d == 'r' then state.has_r = true end
8308             state.sim = state.sim or item
8309             state.eim = item
8310         elseif d == 'l' and state.sim and state.has_r then
8311             head, state = insert_implicit(head, state, outer)
8312         elseif d == 'l' then
8313             state.sim, state.eim, state.has_r = nil, nil, false
8314         end
8315     else
8316         if d == 'an' or d == 'l' then
8317             if nodes[q][3] then -- nil except after an explicit dir
8318                 state.sim = item -- so we move sim 'inside' the group
8319             else
8320                 state.sim = state.sim or item
8321             end
8322             state.eim = item
8323         elseif d == 'r' and state.sim then
8324             head, state = insert_implicit(head, state, outer)
8325         elseif d == 'r' then
8326             state.sim, state.eim = nil, nil
8327         end
8328     end
8329
8330     if isdir then
8331         last = d          -- Don't search back - best save now
8332     elseif d == 'on' and state.san then
8333         state.san = state.san or item
8334         state.ean = item
8335     end
8336
8337 end
8338
8339 head = node.prev(head) or head

```

```

8340 % \end{macrocode}
8341 %
8342 % Now direction nodes has been distributed with relation to characters
8343 % and spaces, we need to take into account \TeX-specific elements in
8344 % the node list, to move them at an appropriate place. Firstly, with
8345 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8346 % that the latter are still discardable.
8347 %
8348 % \begin{macrocode}
8349 --- FIXES ---
8350 if has_hyperlink then
8351   local flag, linking = 0, 0
8352   for item in node.traverse(head) do
8353     if item.id == DIR then
8354       if item.dir == '+TRT' or item.dir == '+TLT' then
8355         flag = flag + 1
8356       elseif item.dir == '-TRT' or item.dir == '-TLT' then
8357         flag = flag - 1
8358       end
8359     elseif item.id == 8 and item.subtype == 19 then
8360       linking = flag
8361     elseif item.id == 8 and item.subtype == 20 then
8362       if linking > 0 then
8363         if item.prev.id == DIR and
8364            (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8365           d = node.new(DIR)
8366           d.dir = item.prev.dir
8367           node.remove(head, item.prev)
8368           node.insert_after(head, item, d)
8369         end
8370       end
8371       linking = 0
8372     end
8373   end
8374 end
8375
8376 for item in node.traverse_id(10, head) do
8377   local p = item
8378   local flag = false
8379   while p.prev and p.prev.id == 14 do
8380     flag = true
8381     p = p.prev
8382   end
8383   if flag then
8384     node.insert_before(head, p, node.copy(item))
8385     node.remove(head, item)
8386   end
8387 end
8388
8389 return head
8390 end
8391
8391 function Babel.unset_atdir(head)
8392   local ATDIR = Babel.attr_dir
8393   for item in node.traverse(head) do
8394     node.set_attribute(item, ATDIR, 0x80)
8395   end
8396   return head
8397 end
8398 /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.

```
8399 (*nil)
8400 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8401 \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.

```
8402 \ifx\l@nil\undefined
8403 \newlanguage\l@nil
8404 \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
8405 \let\bbl@elt\relax
8406 \edef\bbl@languages{% Add it to the list of languages
8407 \bbl@languages\bbl@elt{nil}{the\l@nil}{}}
8408 \fi
```

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

```
8409 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

`\captionnil`
`\datenil`

```
8410 \let\captionnil\@empty
8411 \let\datenil\@empty
```

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

```
8412 \def\bbl@inidata@nil{%
8413 \bbl@elt{identification}{tag.ini}{und}%
8414 \bbl@elt{identification}{load.level}{0}%
8415 \bbl@elt{identification}{charset}{utf8}%
8416 \bbl@elt{identification}{version}{1.0}%
8417 \bbl@elt{identification}{date}{2022-05-16}%
8418 \bbl@elt{identification}{name.local}{nil}%
8419 \bbl@elt{identification}{name.english}{nil}%
8420 \bbl@elt{identification}{name.babel}{nil}%
8421 \bbl@elt{identification}{tag.bcp47}{und}%
8422 \bbl@elt{identification}{language.tag.bcp47}{und}%
8423 \bbl@elt{identification}{tag.opentype}{dfLT}%
8424 \bbl@elt{identification}{script.name}{Latin}%
8425 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8426 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8427 \bbl@elt{identification}{level}{1}%
```

```

8428 \bbl@elt{identification}{encodings}{}%
8429 \bbl@elt{identification}{derivate}{no}}
8430 \@namedef{bbl@tbc@nil}{und}
8431 \@namedef{bbl@lbc@nil}{und}
8432 \@namedef{bbl@casing@nil}{und}
8433 \@namedef{bbl@lotf@nil}{dflt}
8434 \@namedef{bbl@elname@nil}{nil}
8435 \@namedef{bbl@lname@nil}{nil}
8436 \@namedef{bbl@esname@nil}{Latin}
8437 \@namedef{bbl@sname@nil}{Latin}
8438 \@namedef{bbl@sbc@nil}{Latn}
8439 \@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.

```

8440 \ldf@finish{nil}
8441 </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.

```

8442 <<*Compute Julian day>> ≡
8443 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8444 \def\bbl@cs@gregleap#1{%
8445   (\bbl@fpmo{#1}{4} == 0) &&
8446   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
8447 \def\bbl@cs@jd#1#2#3{% year, month, day
8448   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8449     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8450     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8451     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8452 <</Compute Julian day>>

```

13.1. Islamic

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

```

8453 <*ca-islamic>
8454 \ExplSyntaxOn
8455 <@Compute Julian day>
8456 % == islamic (default)
8457 % Not yet implemented
8458 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8459 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8460   ((#3 + ceil(29.5 * (#2 - 1)) +
8461     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8462     1948439.5) - 1) }
8463 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8464 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8465 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8466 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8467 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8468 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8469   \edef\bbl@tempa{%
8470     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 }%
8471   }
8472   \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) } }%
8473   \edef#6{\fp_eval:n{

```

```

8474 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }%
8475 \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).

```

8476 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8477 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8478 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8479 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8480 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8481 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8482 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8483 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8484 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8485 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8486 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8487 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8488 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8489 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8490 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8491 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8492 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8493 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8494 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8495 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8496 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8497 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8498 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8499 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8500 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8501 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8502 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8503 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8504 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8505 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8506 65401,65431,65460,65490,65520}
8507 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8508 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8509 \namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8510 \def\bbl@ca@islamcuqr@x#1#2-#3-#4@@#5#6#7{%
8511 \ifnum#2>2014 \ifnum#2<2038
8512 \bbl@afterfi\expandafter\@gobble
8513 \fi\fi
8514 {\bbl@error{year-out-range}{2014-2038}}}%
8515 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8516 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8517 \count@\@ne
8518 \bbl@foreach\bbl@cs@umalqura@data{%
8519 \advance\count@\@ne
8520 \ifnum##1>\bbl@tempd\else
8521 \edef\bbl@tempe{\the\count@}%
8522 \edef\bbl@tempb{##1}%
8523 \fi}%
8524 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
8525 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8526 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8527 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8528 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8529 \ExplSyntaxOff
8530 \bbl@add\bbl@precalendar{%
8531 \bbl@replace\bbl@ld@calendar{-civil}}}%

```

```

8532 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
8533 \bbl@replace\bbl@ld@calendar{+}{}%
8534 \bbl@replace\bbl@ld@calendar{-}{%}
8535 </ca-islamic>

```

13.2. Hebrew

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

```

8536 <{*ca-hebrew}>
8537 \newcount\bbl@cntcommon
8538 \def\bbl@remainder#1#2#3{%
8539   #3=#1\relax
8540   \divide #3 by #2\relax
8541   \multiply #3 by -#2\relax
8542   \advance #3 by #1\relax}%
8543 \newif\ifbbl@divisible
8544 \def\bbl@checkifdivisible#1#2{%
8545   {\countdef\tmp=0
8546    \bbl@remainder{#1}{#2}{\tmp}%
8547    \ifnum \tmp=0
8548      \global\bbl@divisibletrue
8549    \else
8550      \global\bbl@divisiblefalse
8551    \fi}}
8552 \newif\ifbbl@gregleap
8553 \def\bbl@ifgregleap#1{%
8554   \bbl@checkifdivisible{#1}{4}%
8555   \ifbbl@divisible
8556     \bbl@checkifdivisible{#1}{100}%
8557     \ifbbl@divisible
8558       \bbl@checkifdivisible{#1}{400}%
8559       \ifbbl@divisible
8560         \bbl@gregleaptrue
8561       \else
8562         \bbl@gregleapfalse
8563       \fi
8564     \else
8565       \bbl@gregleaptrue
8566     \fi
8567   \else
8568     \bbl@gregleapfalse
8569   \fi
8570   \ifbbl@gregleap}
8571 \def\bbl@gregdayspriormonths#1#2#3{%
8572   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8573     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8574   \bbl@ifgregleap{#2}%
8575   \ifnum #1 > 2
8576     \advance #3 by 1
8577   \fi
8578   \fi
8579   \global\bbl@cntcommon=#3}%
8580   #3=\bbl@cntcommon}
8581 \def\bbl@gregdaysprioryears#1#2{%
8582   {\countdef\tmpc=4
8583    \countdef\tmpb=2
8584    \tmpb=#1\relax
8585    \advance \tmpb by -1
8586    \tmpc=\tmpb
8587    \multiply \tmpc by 365
8588    #2=\tmpc

```

```

8589 \tmpc=\tmpb
8590 \divide \tmpc by 4
8591 \advance #2 by \tmpc
8592 \tmpc=\tmpb
8593 \divide \tmpc by 100
8594 \advance #2 by -\tmpc
8595 \tmpc=\tmpb
8596 \divide \tmpc by 400
8597 \advance #2 by \tmpc
8598 \global\bbl@cntcommon=#2\relax}%
8599 #2=\bbl@cntcommon}
8600 \def\bbl@absfromgreg#1#2#3#4{%
8601 {\countdef\tmpd=0
8602 #4=#1\relax
8603 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8604 \advance #4 by \tmpd
8605 \bbl@gregdaysprioryears{#3}{\tmpd}%
8606 \advance #4 by \tmpd
8607 \global\bbl@cntcommon=#4\relax}%
8608 #4=\bbl@cntcommon}
8609 \newif\ifbbl@hebrleap
8610 \def\bbl@checkleaphebrewyear#1{%
8611 {\countdef\tmpa=0
8612 \countdef\tmpb=1
8613 \tmpa=#1\relax
8614 \multiply \tmpa by 7
8615 \advance \tmpa by 1
8616 \bbl@remainder{\tmpa}{19}{\tmpb}%
8617 \ifnum \tmpb < 7
8618 \global\bbl@hebrleaptrue
8619 \else
8620 \global\bbl@hebrleapfalse
8621 \fi}}
8622 \def\bbl@hebreleapsedmonths#1#2{%
8623 {\countdef\tmpa=0
8624 \countdef\tmpb=1
8625 \countdef\tmpc=2
8626 \tmpa=#1\relax
8627 \advance \tmpa by -1
8628 #2=\tmpa
8629 \divide #2 by 19
8630 \multiply #2 by 235
8631 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8632 \tmpc=\tmpb
8633 \multiply \tmpb by 12
8634 \advance #2 by \tmpb
8635 \multiply \tmpc by 7
8636 \advance \tmpc by 1
8637 \divide \tmpc by 19
8638 \advance #2 by \tmpc
8639 \global\bbl@cntcommon=#2}%
8640 #2=\bbl@cntcommon}
8641 \def\bbl@hebreleapseddays#1#2{%
8642 {\countdef\tmpa=0
8643 \countdef\tmpb=1
8644 \countdef\tmpc=2
8645 \bbl@hebreleapsedmonths{#1}{#2}%
8646 \tmpa=#2\relax
8647 \multiply \tmpa by 13753
8648 \advance \tmpa by 5604
8649 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8650 \divide \tmpa by 25920
8651 \multiply #2 by 29

```

```

8652 \advance #2 by 1
8653 \advance #2 by \tmpa
8654 \bbl@remainder{#2}{7}{\tmpa}%
8655 \ifnum \tmpc < 19440
8656 \ifnum \tmpc < 9924
8657 \else
8658 \ifnum \tmpa=2
8659 \bbl@checkleaphebrewyear{#1}% of a common year
8660 \ifbbl@hebrleap
8661 \else
8662 \advance #2 by 1
8663 \fi
8664 \fi
8665 \fi
8666 \ifnum \tmpc < 16789
8667 \else
8668 \ifnum \tmpa=1
8669 \advance #1 by -1
8670 \bbl@checkleaphebrewyear{#1}% at the end of leap year
8671 \ifbbl@hebrleap
8672 \advance #2 by 1
8673 \fi
8674 \fi
8675 \fi
8676 \else
8677 \advance #2 by 1
8678 \fi
8679 \bbl@remainder{#2}{7}{\tmpa}%
8680 \ifnum \tmpa=0
8681 \advance #2 by 1
8682 \else
8683 \ifnum \tmpa=3
8684 \advance #2 by 1
8685 \else
8686 \ifnum \tmpa=5
8687 \advance #2 by 1
8688 \fi
8689 \fi
8690 \fi
8691 \global\bbl@cntcommon=#2\relax}%
8692 #2=\bbl@cntcommon}
8693 \def\bbl@daysinhebrewyear#1#2{%
8694 {\countdef\tmpe=12
8695 \bbl@hebreleapseddays{#1}{\tmpe}%
8696 \advance #1 by 1
8697 \bbl@hebreleapseddays{#1}{#2}%
8698 \advance #2 by -\tmpe
8699 \global\bbl@cntcommon=#2}%
8700 #2=\bbl@cntcommon}
8701 \def\bbl@hebrdayspriormonths#1#2#3{%
8702 {\countdef\tmpf= 14
8703 #3=\ifcase #1
8704 0 \or
8705 0 \or
8706 30 \or
8707 59 \or
8708 89 \or
8709 118 \or
8710 148 \or
8711 148 \or
8712 177 \or
8713 207 \or
8714 236 \or

```

```

8715         266 \or
8716         295 \or
8717         325 \or
8718         400
8719     \fi
8720     \bbl@checkleaphebryear{#2}%
8721     \ifbbl@hebrleap
8722         \ifnum #1 > 6
8723             \advance #3 by 30
8724         \fi
8725     \fi
8726     \bbl@daysinhebryear{#2}{\tmpf}%
8727     \ifnum #1 > 3
8728         \ifnum \tmpf=353
8729             \advance #3 by -1
8730         \fi
8731         \ifnum \tmpf=383
8732             \advance #3 by -1
8733         \fi
8734     \fi
8735     \ifnum #1 > 2
8736         \ifnum \tmpf=355
8737             \advance #3 by 1
8738         \fi
8739         \ifnum \tmpf=385
8740             \advance #3 by 1
8741         \fi
8742     \fi
8743     \global\bbl@cntcommon=#3\relax}%
8744     #3=\bbl@cntcommon}
8745 \def\bbl@absfromhebr#1#2#3#4{%
8746     {#4=#1\relax
8747     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8748     \advance #4 by #1\relax
8749     \bbl@hebrrelapseddays{#3}{#1}%
8750     \advance #4 by #1\relax
8751     \advance #4 by -1373429
8752     \global\bbl@cntcommon=#4\relax}%
8753     #4=\bbl@cntcommon}
8754 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8755     {\countdef\tmpx= 17
8756     \countdef\tmpy= 18
8757     \countdef\tmpz= 19
8758     #6=#3\relax
8759     \global\advance #6 by 3761
8760     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8761     \tmpz=1 \tmpy=1
8762     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8763     \ifnum \tmpx > #4\relax
8764         \global\advance #6 by -1
8765         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8766     \fi
8767     \advance #4 by -\tmpx
8768     \advance #4 by 1
8769     #5=#4\relax
8770     \divide #5 by 30
8771     \loop
8772         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8773         \ifnum \tmpx < #4\relax
8774             \advance #5 by 1
8775             \tmpy=\tmpx
8776         \repeat
8777     \global\advance #5 by -1

```

```

8778 \global\advance #4 by -\tmpy}}
8779 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8780 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8781 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8782 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8783 \bbl@hebrfromgreg
8784 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8785 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8786 \edef#4{\the\bbl@hebryear}%
8787 \edef#5{\the\bbl@hebrmonth}%
8788 \edef#6{\the\bbl@hebrday}}
8789 </ca-hebrew>

```

13.3. Persian

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

```

8790 <*ca-persian>
8791 \ExplSyntaxOn
8792 <@Compute Julian day@>
8793 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8794 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8795 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8796 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8797 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8798 \bbl@afterfi\expandafter\@gobble
8799 \fi\fi
8800 {\bbl@error{year-out-range}{2013-2050}{}}}%
8801 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8802 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8803 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8804 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8805 \ifnum\bbl@tempc<\bbl@tempb
8806 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8807 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8808 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8809 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8810 \fi
8811 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8812 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8813 \edef#5{\fp_eval:n{% set Jalali month
8814 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8815 \edef#6{\fp_eval:n{% set Jalali day
8816 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
8817 \ExplSyntaxOff
8818 </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.

```

8819 <*ca-coptic>
8820 \ExplSyntaxOn
8821 <@Compute Julian day@>
8822 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8823 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8824 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8825 \edef#4{\fp_eval:n{%
8826 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%

```



```

8827 \edef\bbl@tempc{\fp_eval:n{%
8828     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8829 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8830 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8831 \ExplSyntaxOff
8832 </ca-coptic>
8833 <*ca-ethiopic>
8834 \ExplSyntaxOn
8835 <@Compute Julian day@>
8836 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8837     \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8838     \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8839     \edef#4{\fp_eval:n{%
8840         floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8841     \edef\bbl@tempc{\fp_eval:n{%
8842         \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8843     \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8844     \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8845 \ExplSyntaxOff
8846 </ca-ethiopic>

```

13.5. Buddhist

That's very simple.

```

8847 <*ca-buddhist>
8848 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8849     \edef#4{\number\numexpr#1+543\relax}%
8850     \edef#5{#2}%
8851     \edef#6{#3}}
8852 </ca-buddhist>
8853 %
8854 % \subsection{Chinese}
8855 %
8856 % Brute force, with the Julian day of first day of each month. The
8857 % table has been computed with the help of \textsf{python-lunardate} by
8858 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8859 % is 2015-2044.
8860 %
8861 % \begin{macrocode}
8862 <*ca-chinese>
8863 \ExplSyntaxOn
8864 <@Compute Julian day@>
8865 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
8866     \edef\bbl@tempd{\fp_eval:n{%
8867         \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8868     \count@ \z@
8869     \@tempcnta=2015
8870     \bbl@foreach\bbl@cs@chinese@data{%
8871         \ifnum##1>\bbl@tempd\else
8872             \advance\count@\@ne
8873             \ifnum\count@>12
8874                 \count@\@ne
8875                 \advance\@tempcnta\@ne\fi
8876             \bbl@xin@{,##1,},{,\bbl@cs@chinese@leap,}%
8877             \ifin@
8878                 \advance\count@\m@ne
8879             \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8880             \else
8881                 \edef\bbl@tempe{\the\count@}%
8882             \fi
8883             \edef\bbl@tempb{##1}%
8884             \fi}%
8885     \edef#4{\the\@tempcnta}%

```

```

8886 \edef#5{\bbl@tempe}%
8887 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8888 \def\bbl@cs@chinese@leap{%
8889 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8890 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8891 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8892 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8893 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8894 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8895 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8896 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8897 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8898 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8899 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8900 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8901 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8902 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8903 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8904 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8905 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8906 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8907 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8908 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8909 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8910 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8911 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8912 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8913 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8914 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8915 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8916 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8917 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8918 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8919 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8920 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8921 10896,10926,10956,10986,11015,11045,11074,11103}
8922 \ExplSyntaxOff
8923 </ca-chinese>

```

14. Support for Plain T_EX (plain.def)

14.1. Not renaming hyphen.tex

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

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

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

The files `bplain.tex` and `lplain.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 `lplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

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

```

8924 <{*bplain | bplain}
8925 \catcode`\{=1 % left brace is begin-group character
8926 \catcode`\}=2 % right brace is end-group character
8927 \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).

```
8928 \openin 0 hyphen.cfg
8929 \ifeof0
8930 \else
8931 \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.

```
8932 \def\input #1 {%
8933 \let\input\input
8934 \a hyphen.cfg
8935 \let\input\undefined
8936 }
8937 \fi
8938 </bplain | bplain>
```

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

```
8939 <bplain>\a plain.tex
8940 <bplain>\a lplain.tex
```

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

```
8941 <bplain>\def\fmtname{babel-plain}
8942 <bplain>\def\fmtname{babel-lplain}
```

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

14.2. Emulating some \LaTeX features

The file `babel.def` expects some definitions made in the $\text{\LaTeX} 2_{\epsilon}$ style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading `babel`. `\BabelModifiers` can be set too (but not sure it works).

```
8943 <<*Emulate LaTeX>> ≡
8944 \def\@empty{}
8945 \def\loadlocalcfg#1{%
8946 \openin0#1.cfg
8947 \ifeof0
8948 \closein0
8949 \else
8950 \closein0
8951 {\immediate\write16{*****}%
8952 \immediate\write16{* Local config file #1.cfg used}%
8953 \immediate\write16{*}%
8954 }
8955 \input #1.cfg\relax
8956 \fi
8957 \@endofldef}
```

14.3. General tools

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

```
8958 \long\def\@firstofone#1{#1}
8959 \long\def\@firstoftwo#1#2{#1}
8960 \long\def\@secondoftwo#1#2{#2}
8961 \def\@nnil{\nil}
8962 \def\@gobbletwo#1#2{}
8963 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
```

```

8964 \def\@star@or@long#1{%
8965   \ifstar
8966     {\let\l@ngrel@x\relax#1}%
8967     {\let\l@ngrel@x\long#1}}
8968 \let\l@ngrel@x\relax
8969 \def\@car#1#2\@nil{#1}
8970 \def\@cdr#1#2\@nil{#2}
8971 \let\@typeset@protect\relax
8972 \let\protected@edef\edef
8973 \long\def\@gobble#1{}
8974 \edef\@backslashchar{\expandafter\@gobble\string\}
8975 \def\strip@prefix#1>{}
8976 \def\g@addto@macro#1#2{%
8977   \toks@\expandafter{#1#2}%
8978   \xdef#1{\the\toks@}}
8979 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8980 \def\@nameuse#1{\csname #1\endcsname}
8981 \def\@ifundefined#1{%
8982   \expandafter\ifx\csname#1\endcsname\relax
8983     \expandafter\@firstoftwo
8984   \else
8985     \expandafter\@secondoftwo
8986   \fi}
8987 \def\@expandtwoargs#1#2#3{%
8988   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8989 \def\zap@space#1 #2{%
8990   #1%
8991   \ifx#2\@empty\else\expandafter\zap@space\fi
8992   #2}
8993 \let\bbl@trace\@gobble
8994 \def\bbl@error#1{% Implicit #2#3#4
8995   \begingroup
8996     \catcode`\=0 \catcode`\==12 \catcode`\`=12
8997     \catcode`\^M=5 \catcode`\%=14
8998     \input errbabel.def
8999   \endgroup
9000   \bbl@error{#1}}
9001 \def\bbl@warning#1{%
9002   \begingroup
9003     \newlinechar=`^^J
9004     \def\{^^J(babel) }%
9005     \message{\{#1}%
9006   \endgroup}
9007 \let\bbl@infowarn\bbl@warning
9008 \def\bbl@info#1{%
9009   \begingroup
9010     \newlinechar=`^^J
9011     \def\{^^J}%
9012     \wlog{#1}%
9013   \endgroup}

```

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

```

9014 \ifx\@preamblecmds\undefined
9015   \def\@preamblecmds{}
9016 \fi
9017 \def\@onlypreamble#1{%
9018   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9019     \@preamblecmds\do#1}}
9020 \@onlypreamble\@onlypreamble

```

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

```

9021 \def\begindocument{%
9022   \@begindocumenthook

```

```

9023 \global\let\@begindocumenthook\@undefined
9024 \def\do##1{\global\let##1\@undefined}%
9025 \@preamblecmds
9026 \global\let\do\noexpand}

9027 \ifx\@begindocumenthook\@undefined
9028 \def\@begindocumenthook{}
9029 \fi
9030 \@onlypreamble\@begindocumenthook
9031 \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.

```

9032 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
9033 \@onlypreamble\AtEndOfPackage
9034 \def\@endoflfd{}
9035 \@onlypreamble\@endoflfd
9036 \let\bbl@afterlang\@empty
9037 \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.

```

9038 \catcode`\&=\z@
9039 \ifx&\if@files\@undefined
9040 \expandafter\let\csname if@files\expandafter\endcsname
9041 \csname iffalse\endcsname
9042 \fi
9043 \catcode`\&=4

```

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

```

9044 \def\newcommand{\@star@or@long\new@command}
9045 \def\new@command#1{%
9046 \testopt{\@newcommand#1}0}
9047 \def\@newcommand#1[#2]{%
9048 \ifnextchar [{\@xargdef#1[#2]}%
9049 {\@argdef#1[#2]}}
9050 \long\def\@argdef#1[#2]#3{%
9051 \@yargdef#1\@ne{#2}{#3}}
9052 \long\def\@xargdef#1[#2][#3]#4{%
9053 \expandafter\def\expandafter#1\expandafter{%
9054 \expandafter\@protected@testopt\expandafter #1%
9055 \csname\string#1\expandafter\endcsname{#3}}%
9056 \expandafter\@yargdef \csname\string#1\endcsname
9057 \tw@{#2}{#4}}
9058 \long\def\@yargdef#1#2#3{%
9059 \@tempcnta#3\relax
9060 \advance \@tempcnta \@ne
9061 \let\@hash@\relax
9062 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9063 \@tempcntb #2%
9064 \@whilenum\@tempcntb <\@tempcnta
9065 \do{%
9066 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
9067 \advance\@tempcntb \@ne}%
9068 \let\@hash@##%
9069 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9070 \def\providecommand{\@star@or@long\provide@command}
9071 \def\provide@command#1{%
9072 \begingroup
9073 \escapechar\m@ne\def\@gtempa{\string#1}%
9074 \endgroup
9075 \expandafter\@ifundefined\@gtempa
9076 {\def\reserved@a{\new@command#1}}%

```

```

9077     {\let\reserved@a\relax
9078     \def\reserved@a{\newcommand\reserved@a}}}%
9079     \reserved@a}%

9080 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9081 \def\declare@robustcommand#1{%
9082     \edef\reserved@a{\string#1}%
9083     \def\reserved@b{#1}%
9084     \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9085     \edef#1{%
9086         \ifx\reserved@a\reserved@b
9087             \noexpand\x@protect
9088             \noexpand#1%
9089         \fi
9090         \noexpand\protect
9091         \expandafter\noexpand\csname
9092             \expandafter\@gobble\string#1 \endcsname
9093     }%
9094     \expandafter\newcommand\csname
9095         \expandafter\@gobble\string#1 \endcsname
9096 }
9097 \def\x@protect#1{%
9098     \ifx\protect\@typeset@protect\else
9099         \x@protect#1%
9100     \fi
9101 }
9102 \catcode`\&=\z@ % Trick to hide conditionals
9103 \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`.

```

9104 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9105 \catcode`\&=4
9106 \ifx\in@\@undefined
9107     \def\in@#1#2{%
9108         \def\in@@##1##2##3\in@@{%
9109             \ifx\in@@##2\in@false\else\in@true\fi}%
9110         \in@@##2#1\in@\in@@}
9111 \else
9112     \let\bbl@tempa\@empty
9113 \fi
9114 \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).

```

9115 \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.

```

9116 \def\@ifl@aded#1#2#3#4{}

```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their $\TeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

9117 \ifx\@tempcnta\@undefined
9118     \csname newcount\endcsname\@tempcnta\relax
9119 \fi
9120 \ifx\@tempcntb\@undefined
9121     \csname newcount\endcsname\@tempcntb\relax
9122 \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`).

```

9123 \ifx\bye\@undefined
9124   \advance\count10 by -2\relax
9125 \fi
9126 \ifx\@ifnextchar\@undefined
9127   \def\@ifnextchar#1#2#3{%
9128     \let\reserved@d=#1%
9129     \def\reserved@a{#2}\def\reserved@b{#3}%
9130     \futurelet\@let@token\@ifnch}
9131   \def\@ifnch{%
9132     \ifx\@let@token\@sptoken
9133       \let\reserved@c\@xifnch
9134     \else
9135       \ifx\@let@token\reserved@d
9136         \let\reserved@c\reserved@a
9137       \else
9138         \let\reserved@c\reserved@b
9139       \fi
9140     \fi
9141     \reserved@c}
9142   \def\{\let\@sptoken= } \: % this makes \@sptoken a space token
9143   \def\{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
9144 \fi
9145 \def\@testopt#1#2{%
9146   \@ifnextchar[#{1}{#1[#2]}}
9147 \def\@protected@testopt#1{%
9148   \ifx\protect\@typeset@protect
9149     \expandafter\@testopt
9150   \else
9151     \@x@protect#1%
9152   \fi}
9153 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9154   #2\relax}\fi}
9155 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9156   \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9157 \def\DeclareTextCommand{%
9158   \@dec@text@cmd\providecommand
9159 }
9160 \def\ProvideTextCommand{%
9161   \@dec@text@cmd\providecommand
9162 }
9163 \def\DeclareTextSymbol#1#2#3{%
9164   \@dec@text@cmd\chardef#1{#2}#3\relax
9165 }
9166 \def\@dec@text@cmd#1#2#3{%
9167   \expandafter\def\expandafter#2%
9168     \expandafter{%
9169       \csname#3-cmd\expandafter\endcsname
9170       \expandafter#2%
9171       \csname#3\string#2\endcsname
9172     }%
9173 %   \let\@ifdefinable\@rc@ifdefinable
9174   \expandafter#1\csname#3\string#2\endcsname
9175 }
9176 \def\@current@cmd#1{%
9177   \ifx\protect\@typeset@protect\else
9178     \noexpand#1\expandafter\@gobble

```

```

9179 \fi
9180 }
9181 \def\@changed@cmd#1#2{%
9182 \ifx\protect\@typeset@protect
9183 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9184 \expandafter\ifx\csname ?\string#1\endcsname\relax
9185 \expandafter\def\csname ?\string#1\endcsname{%
9186 \@changed@x@err{#1}%
9187 }%
9188 \fi
9189 \global\expandafter\let
9190 \csname\cf@encoding\string#1\expandafter\endcsname
9191 \csname ?\string#1\endcsname
9192 \fi
9193 \csname\cf@encoding\string#1%
9194 \expandafter\endcsname
9195 \else
9196 \noexpand#1%
9197 \fi
9198 }
9199 \def\@changed@x@err#1{%
9200 \errhelp{Your command will be ignored, type <return> to proceed}%
9201 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9202 \def\DeclareTextCommandDefault#1{%
9203 \DeclareTextCommand#1?%
9204 }
9205 \def\ProvideTextCommandDefault#1{%
9206 \ProvideTextCommand#1?%
9207 }
9208 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9209 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9210 \def\DeclareTextAccent#1#2#3{%
9211 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9212 }
9213 \def\DeclareTextCompositeCommand#1#2#3#4{%
9214 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9215 \edef\reserved@b{\string##1}%
9216 \edef\reserved@c{%
9217 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9218 \ifx\reserved@b\reserved@c
9219 \expandafter\expandafter\expandafter\ifx
9220 \expandafter\@car\reserved@a\relax\relax\@nil
9221 \@text@composite
9222 \else
9223 \edef\reserved@b##1{%
9224 \def\expandafter\noexpand
9225 \csname#2\string#1\endcsname###1{%
9226 \noexpand\@text@composite
9227 \expandafter\noexpand\csname#2\string#1\endcsname
9228 ###1\noexpand\@empty\noexpand\@text@composite
9229 {##1}%
9230 }%
9231 }%
9232 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9233 \fi
9234 \expandafter\def\csname\expandafter\string\csname
9235 #2\endcsname\string#1-\string#3\endcsname{#4}
9236 \else
9237 \errhelp{Your command will be ignored, type <return> to proceed}%
9238 \errmessage{\string\DeclareTextCompositeCommand\space used on
9239 inappropriate command \protect#1}
9240 \fi
9241 }

```



```

9242 \def\@text@composite#1#2#3\@text@composite{%
9243   \expandafter\@text@composite@x
9244   \csname\string#1-\string#2\endcsname
9245 }
9246 \def\@text@composite@x#1#2{%
9247   \ifx#1\relax
9248     #2%
9249   \else
9250     #1%
9251   \fi
9252 }
9253 %
9254 \def\@strip@args#1:#2-#3\@strip@args{#2}
9255 \def\DeclareTextComposite#1#2#3#4{%
9256   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9257   \bgroup
9258     \lccode\@=#4%
9259     \lowercase{%
9260   \egroup
9261     \reserved@a \@%
9262   }%
9263 }
9264 %
9265 \def\UseTextSymbol#1#2{#2}
9266 \def\UseTextAccent#1#2#3{}
9267 \def\@use@text@encoding#1{}
9268 \def\DeclareTextSymbolDefault#1#2{%
9269   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9270 }
9271 \def\DeclareTextAccentDefault#1#2{%
9272   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9273 }
9274 \def\cf@encoding{OT1}

```

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

```

9275 \DeclareTextAccent{"}{OT1}{127}
9276 \DeclareTextAccent{'}{OT1}{19}
9277 \DeclareTextAccent{^}{OT1}{94}
9278 \DeclareTextAccent{\`}{OT1}{18}
9279 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9280 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9281 \DeclareTextSymbol{\textquotedblright}{OT1}{'\'}
9282 \DeclareTextSymbol{\textquoteleft}{OT1}{'\'}
9283 \DeclareTextSymbol{\textquoteright}{OT1}{'\'}
9284 \DeclareTextSymbol{\i}{OT1}{16}
9285 \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`.

```

9286 \ifx\scriptsize\undefined
9287   \let\scriptsize\sevenrm
9288 \fi

```

And a few more “dummy” definitions.

```

9289 \def\language{english}%
9290 \let\bbl@opt@shorthands\@nnil
9291 \def\bbl@ifshorthand#1#2#3#2{%
9292   \let\bbl@language@opts\@empty
9293   \let\bbl@provide@locale\relax
9294   \ifx\babeloptionstrings\undefined
9295     \let\bbl@opt@strings\@nnil

```

```

9296 \else
9297 \let\bbl@opt@strings\babeloptionstrings
9298 \fi
9299 \def\BabelStringsDefault{generic}
9300 \def\bbl@tempa{normal}
9301 \ifx\babeloptionmath\bbl@tempa
9302 \def\bbl@mathnormal{\noexpand\textormath}
9303 \fi
9304 \def\AfterBabelLanguage#1#2{}
9305 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9306 \let\bbl@afterlang\relax
9307 \def\bbl@opt@safe{BR}
9308 \ifx\@uclclist\undefined\let\@uclclist\empty\fi
9309 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9310 \expandafter\newif\csname ifbbl@single\endcsname
9311 \chardef\bbl@bidimode\z@
9312 <</Emulate LaTeX>>

A proxy file:

9313 <*plain>
9314 \input babel.def
9315 </plain>

```

15. Acknowledgements

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

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

Barbara Beeton has helped in improving the manual.

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

References

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