

Babel

Code

Version 25.15.105483
2025/11/19

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

LuaT_EX

pdfT_EX

XeT_EX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	A few core definitions	8
3.2	LaTeX: babel.sty (start)	8
3.3	base	9
3.4	key=value options and other general option	10
3.5	Post-process some options	11
3.6	Plain: babel.def (start)	13
4	babel.sty and babel.def (common)	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	More on selection	24
4.4	Short tags	25
4.5	Compatibility with language.def	25
4.6	Hooks	26
4.7	Setting up language files	27
4.8	Shorthands	29
4.9	Language attributes	38
4.10	Support for saving and redefining macros	39
4.11	French spacing	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’	62
4.20	Processing keys in ini	66
4.21	French spacing (again)	72
4.22	Handle language system	73
4.23	Numerals	73
4.24	Casing	75
4.25	Getting info	76
4.26	BCP 47 related commands	77
5	Adjusting the Babel behavior	78
5.1	Cross referencing macros	80
5.2	Layout	83
5.3	Marks	84
5.4	Other packages	85
5.4.1	ifthen	85
5.4.2	varioref	86
5.4.3	hhline	86
5.5	Encoding and fonts	87
5.6	Basic bidi support	88
5.7	Local Language Configuration	92
5.8	Language options	92

6	The kernel of Babel	96
7	Error messages	96
8	Loading hyphenation patterns	100
9	luatex + xetex: common stuff	104
10	Hooks for XeTeX and LuaTeX	107
10.1	XeTeX	107
10.2	Support for interchar	109
10.3	Layout	111
10.4	8-bit TeX	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	128
10.11	Bidi	135
10.12	Layout	137
10.13	Lua: transforms	147
10.14	Lua: Auto bidi with basic and basic-r	157
11	Data for CJK	168
12	The ‘nil’ language	168
13	Calendars	169
13.1	Islamic	170
13.2	Hebrew	171
13.3	Persian	175
13.4	Coptic and Ethiopic	176
13.5	Julian	176
13.6	Buddhist	176
14	Support for Plain T_EX (plain.def)	178
14.1	Not renaming hyphen.tex	178
14.2	Emulating some L ^A T _E X features	179
14.3	General tools	179
14.4	Encoding related macros	183
15	Acknowledgements	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=25.15.105483>>
2 <<date=2025/11/19>>
```

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@csarg#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 \ifcename, which is more efficient, and does not waste memory. Defined inside a group, to avoid \ifcename being implicitly set to \relax by the \cename test.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

101 \global\let\bbl@done\empty

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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


```

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 \chardef\bbl@ldfflag\z@
357 \DeclareOption{provide=!}{\chardef\bbl@ldfflag\@ne} % main = 1
358 \DeclareOption{provide+=!}{\chardef\bbl@ldfflag\tw@} % second = 2
359 \DeclareOption{provide*=!}{\chardef\bbl@ldfflag\thr@@} % second + main
360 % Don't use. Experimental.
361 \newif\ifbbl@single
362 \DeclareOption{selectors=off}{\bbl@singletrue}
363 <@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.

```

364 \let\bbl@opt@shorthands\@nnil
365 \let\bbl@opt@config\@nnil
366 \let\bbl@opt@main\@nnil
367 \let\bbl@opt@headfoot\@nnil
368 \let\bbl@opt@layout\@nnil
369 \let\bbl@opt@provide\@nnil

```

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

```

370 \def\bbl@tempa#1=#2\bbl@tempa{%
371   \bbl@csarg\ifx{opt@#1}\@nnil
372   \bbl@csarg\edef{opt@#1}{#2}%
373   \else
374   \bbl@error{bad-package-option}{#1}{#2}{}%
375   \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.

```

376 \let\bbl@language@opts\@empty
377 \DeclareOption*{%
378   \bbl@xin@{\string=}{\CurrentOption}%
379   \ifin@
380     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
381   \else
382     \bbl@add@list\bbl@language@opts{\CurrentOption}%
383   \fi}

```

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

```

384 \ProcessOptions*

```

3.5. Post-process some options

```

385 \ifx\bbl@opt@provide\@nnil
386   \let\bbl@opt@provide\@empty % %%% MOVE above
387 \else
388   \chardef\bbl@iniflag\@ne
389   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%

```

```

390 \in@{,provide,}{,#1,}%
391 \ifin@
392 \def\bbl@opt@provide{#2}%
393 \fi}
394 \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=...`

```

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

```

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

```

409 \def\bbl@ifshorthand#1{%
410 \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
411 \ifin@
412 \expandafter\@firstoftwo
413 \else
414 \expandafter\@secondoftwo
415 \fi}

```

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

```

416 \edef\bbl@opt@shorthands{%
417 \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.

```

418 \bbl@ifshorthand{'}%
419 {\PassOptionsToPackage{activeacute}{babel}}{}
420 \bbl@ifshorthand{`}%
421 {\PassOptionsToPackage{activegrave}{babel}}{}
422 \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.

```

423 \ifx\bbl@opt@headfoot\@nnil\else
424 \g@addto@macro\@resetactivechars{%
425 \set@typeset@protect
426 \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
427 \let\protect\noexpand}
428 \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.

```

429 \ifx\bbl@opt@safe\@undefined
430 \def\bbl@opt@safe{BR}
431 % \let\bbl@opt@safe\@empty % Pending of \cite
432 \fi

```

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

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

```

433 \bbl@trace{Defining IfBabelLayout}

```

```

434 \ifx\bbl@opt@layout\@nnil
435 \newcommand\IfBabelLayout[3]{#3}%
436 \else
437 \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
438 \in@{,layout,}{, #1,}%
439 \ifin@
440 \def\bbl@opt@layout{#2}%
441 \bbl@replace\bbl@opt@layout{ }{.}%
442 \fi}
443 \newcommand\IfBabelLayout[1]{%
444 \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
445 \ifin@
446 \expandafter\@firstoftwo
447 \else
448 \expandafter\@secondoftwo
449 \fi}
450 \fi
451 \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.

```

452 \ifx\ldf@quit\undefined\else
453 \endinput\fi % Same line!
454 \endinput\fi % Make sure ProvidesFile is defined
455 \ProvidesFile{babel.def}[<@date> v<@version> Babel common definitions]
456 \ifx\AtBeginDocument\undefined
457 \endinput\fi
458 \end{Emulate LaTeX}
459 \fi
460 \end{Basic macros}
461 \end{core}

```

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

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

```

462 \ifx\ldf@quit\undefined\else
463 \endinput\fi % Make sure ProvidesFile is defined
464 \endinput\fi % Make sure ProvidesFile is defined
465 \end{Define core switching macros}

```

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

```

466 \def\adddialect#1#2{%
467 \global\chardef#1#2\relax
468 \bbl@usehooks{adddialect}{#1}{#2}%
469 \begingroup
470 \count@#1\relax
471 \def\bbl@elt##1##2##3##4{%
472 \ifnum\count@=#1\relax
473 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
474 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
475 set to \expandafter\string\csname l@##1\endcsname\%
476 (\string\language\the\count@). Reported}%
477 \def\bbl@elt###1####2####3####4{%
478 \fi}%
479 \bbl@cs{languages}%
480 \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.

```

481 \def\bbl@fixname#1{%
482   \begingroup
483   \def\bbl@tempe{l}%
484   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
485   \bbl@tempd
486     {\lowercase\expandafter{\bbl@tempd}%
487      {\uppercase\expandafter{\bbl@tempd}%
488       \@empty
489        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
490         {\uppercase\expandafter{\bbl@tempd}}}%
491        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
492         {\lowercase\expandafter{\bbl@tempd}}}%
493       \@empty
494       \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
495   \bbl@tempd
496   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}
497 \def\bbl@iflanguage#1{%
498   \@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`.

```

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

```

```

531     \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
532     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}}%
533     {}}%
534     \fi
535     \ifx\bbl@bcp\relax
536         \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
537     \fi
538 \fi\fi}
539 \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.

```

540 \def\iflanguage#1{%
541     \bbl@iflanguage{#1}{%
542         \ifnum\csname l@#1\endcsname=\language
543             \expandafter\@firstoftwo
544         \else
545             \expandafter\@secondoftwo
546         \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.

```

547 \let\bbl@select@type\z@
548 \edef\selectlanguage{%
549     \noexpand\protect
550     \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`.

```

551 \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.

```

552 \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.

```

553 \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:

```

554 \def\bbl@push@language{%
555   \ifx\language\undefined\else
556     \ifx\currentgrouplevel\undefined
557       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
558     \else
559       \ifnum\currentgrouplevel=\z@
560         \xdef\bbl@language@stack{\language+}%
561       \else
562         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
563       \fi
564     \fi
565 \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.

```

566 \def\bbl@pop@lang#1+#2\@@{%
567   \edef\language{#1}%
568   \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).

```

569 \let\bbl@ifrestoring\@secondoftwo
570 \def\bbl@pop@language{%
571   \expandafter\bbl@pop@lang\bbl@language@stack\@@
572   \let\bbl@ifrestoring\@firstoftwo
573   \expandafter\bbl@set@language\expandafter{\language}%
574   \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).

```

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

```

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

```

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

```

```

593 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
594 \bbl@push@language
595 \aftergroup\bbl@pop@language
596 \bbl@set@language{#1}}
597 \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).

```

598 \def\BabelContentsFiles{toc,lof,lot}
599 \def\bbl@set@language#1{% from selectlanguage, pop@
600 % The old buggy way. Preserved for compatibility, but simplified
601 \edef\language{\expandafter\string#1\@empty}%
602 \select@language{\language}%
603 % write to auxs
604 \expandafter\ifx\csname date\language\endcsname\relax\else
605 \if@filesw
606 \ifx\babel@aux@\gobbletwo\else % Set if single in the first, redundant
607 \bbl@savelastskip
608 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
609 \bbl@restorelastskip
610 \fi
611 \bbl@usehooks{write}{}%
612 \fi
613 \fi}
614 %
615 \let\bbl@restorelastskip\relax
616 \let\bbl@savelastskip\relax
617 %
618 \def\select@language#1{% from set@, babel@aux, babel@toc
619 \ifx\bbl@select@name\@empty
620 \def\bbl@select@name{select}%
621 \fi
622 % set hmap
623 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
624 % set name (when coming from babel@aux)
625 \edef\language{#1}%
626 \bbl@fixname\language
627 % define \localename when coming from set@, with a trick
628 \ifx\scantokens\@undefined
629 \def\localename{??}%
630 \else
631 \bbl@exp{\scantokens{\def\localename{\language}\noexpand}\relax}%
632 \fi
633 \bbl@provide@locale
634 \bbl@iflanguage\language{%
635 \let\bbl@select@type\z@
636 \expandafter\bbl@switch\expandafter{\language}}
637 \def\babel@aux#1#2{%
638 \select@language{#1}%
639 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
640 \@writefile{#1}{\babel@toc{#1}{#2}\relax}}}%
641 \def\babel@toc#1#2{%
642 \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`.

```

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

```

```

692 \fi
693 \fi
694 \let\bbl@hymapsel\@cclv
695 % hyphenation - select rules
696 \ifnum\csname l@language\endcsname=\l@unhyphenated
697 \edef\bbl@tempa{u}%
698 \else
699 \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
700 \fi
701 % linebreaking - handle u, e, k (v in the future)
702 \bbl@xin@{/u}{/\bbl@tempa}%
703 \ifin@{\else\bbl@xin@{/e}{/\bbl@tempa}}\fi % elongated forms
704 \ifin@{\else\bbl@xin@{/k}{/\bbl@tempa}}\fi % only kashida
705 \ifin@{\else\bbl@xin@{/p}{/\bbl@tempa}}\fi % padding (e.g., Tibetan)
706 \ifin@{\else\bbl@xin@{/v}{/\bbl@tempa}}\fi % variable font
707 % hyphenation - save mins
708 \babel@savevariable\lefthyphenmin
709 \babel@savevariable\righthyphenmin
710 \ifnum\bbl@engine=\@ne
711 \babel@savevariable\hyphenationmin
712 \fi
713 \ifin@
714 % unhyphenated/kashida/elongated/padding = allow stretching
715 \language\l@unhyphenated
716 \babel@savevariable\emergencystretch
717 \emergencystretch\maxdimen
718 \babel@savevariable\hbadness
719 \hbadness\@M
720 \else
721 % other = select patterns
722 \bbl@patterns{#1}%
723 \fi
724 % hyphenation - set mins
725 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
726 \set@hyphenmins\tw@\thr@@\relax
727 \@nameuse{\bbl@hyphenmins@}%
728 \else
729 \expandafter\expandafter\expandafter\set@hyphenmins
730 \csname #1hyphenmins\endcsname\relax
731 \fi
732 \@nameuse{\bbl@hyphenmins@}%
733 \@nameuse{\bbl@hyphenmins@\language}%
734 \@nameuse{\bbl@hyphenatmin@}%
735 \@nameuse{\bbl@hyphenatmin@\language}%
736 \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.

```

737 \long\def\otherlanguage#1{%
738 \def\bbl@selectorname{other}%
739 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
740 \csname selectlanguage\endcsname{#1}%
741 \ignorespaces}

```

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

```

742 \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`.

```

743 \expandafter\def\csname otherlanguage*\endcsname{%
744   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
745 \def\bbl@otherlanguage@s[#1]#2{%
746   \def\bbl@selectorname{other*}%
747   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
748   \def\bbl@select@opts{#1}%
749   \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”.

```

750 \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.

```

751 \providecommand\bbl@beforeforeign{}
752 \edef\foreignlanguage{%
753   \noexpand\protect
754   \expandafter\noexpand\csname foreignlanguage \endcsname}
755 \expandafter\def\csname foreignlanguage \endcsname{%
756   \@ifstar\bbl@foreign@s\bbl@foreign@x}
757 \providecommand\bbl@foreign@x[3][]{%
758   \begingroup
759     \def\bbl@selectorname{foreign}%
760     \def\bbl@select@opts{#1}%
761     \let\BabelText\@firstofone
762     \bbl@beforeforeign
763     \foreign@language{#2}%
764     \bbl@usehooks{foreign}{}%
765     \BabelText{#3}% Now in horizontal mode!
766   \endgroup}
767 \def\bbl@foreign@s#1#2{%
768   \begingroup
769     {\par}%
770     \def\bbl@selectorname{foreign*}%
771     \let\bbl@select@opts\@empty
772     \let\BabelText\@firstofone
773     \foreign@language{#1}%
774     \bbl@usehooks{foreign*}{}%
775     \bbl@dirparastext
776     \BabelText{#2}% Still in vertical mode!
777     {\par}%
778   \endgroup}
779 \providecommand\BabelWrapText[1]{%
780   \def\bbl@tempa{\def\BabelText###1}%
781   \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.

```

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

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

```

795 \def\IfBabelSelectorTF#1{%
796   \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
797   \ifin@
798     \expandafter\@firstoftwo
799   \else
800     \expandafter\@secondoftwo
801   \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.

```

802 \let\bbl@hyphlist\@empty
803 \let\bbl@hyphenation@\relax
804 \let\bbl@pttnlist\@empty
805 \let\bbl@patterns@\relax
806 \let\bbl@hymapsel=\ccclv
807 \def\bbl@patterns#1{%
808   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
809     \csname l@#1\endcsname
810     \edef\bbl@tempa{#1}%
811   \else
812     \csname l@#1:\f@encoding\endcsname
813     \edef\bbl@tempa{#1:\f@encoding}%
814   \fi
815   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
816   % > luatex
817   \@ifundefined{bbl@hyphenation@}{{}% Can be \relax!
818   \begingroup
819     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
820     \ifin@\else
821       \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
822       \hyphenation{%
823         \bbl@hyphenation@
824         \@ifundefined{bbl@hyphenation@#1}%
825           \@empty
826           {\space\csname bbl@hyphenation@#1\endcsname}}%
827       \xdef\bbl@hyphlist{\bbl@hyphlist\bbl@hyphlist\number\language,}%
828     \fi
829   \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*`.

```

830 \def\hyphenrules#1{%
831   \edef\bbl@tempf{#1}%
832   \bbl@fixname\bbl@tempf
833   \bbl@iflanguage\bbl@tempf{%
834     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
835     \ifx\languageshorthands\@undefined\else
836       \languageshorthands{none}%
837     \fi
838     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
839       \set@hyphenmins\tw@\thr@\relax
840     \else
841       \expandafter\expandafter\expandafter\set@hyphenmins
842       \csname\bbl@tempf hyphenmins\endcsname\relax
843     \fi}}
844 \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.

```

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

```

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

```

849 \def\set@hyphenmins#1#2{%
850   \lefthyphenmin#1\relax
851   \righthyphenmin#2\relax}

```

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

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

```

852 \ifx\ProvidesFile\@undefined
853   \def\ProvidesLanguage#1[#2 #3 #4]{%
854     \wlog{Language: #1 #4 #3 <#2>}%
855   }
856 \else
857   \def\ProvidesLanguage#1{%
858     \begingroup
859     \catcode`\ 10 %
860     \@makeother\%
861     \@ifnextchar[%]
862       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
863   \def\@provideslanguage#1[#2]{%
864     \wlog{Language: #1 #2}%
865     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
866   \endgroup}
867 \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`.

```

868 \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.

```

869 \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’:

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

4.2. Errors

\@nolanerr

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

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

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

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

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

Here ended the now discarded switch.def.

Here also (currently) ends the base option.

```
908 \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.

```

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

```

```

961 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
962 \ifin@else
963 \bbl@tempb##1\@empty
964 \fi
965 \expandafter\bbl@tempa
966 \fi}%
967 \bbl@tempa##1\@empty}
968 \def\bbl@captionslist{%
969 \prefacename\refname\abstractname\bibname\chaptername\appendixname
970 \contentsname\listfigurename\listtablename\indexname\figurename
971 \tablename\partname\enclname\ccname\headtoname\pagename\seename
972 \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.

```

973 \bbl@trace{Short tags}
974 \newcommand\babeltags[1]{%
975 \edef\bbl@tempa{\zap@space#1 \@empty}%
976 \def\bbl@tempb##1=##2\@{ %
977 \edef\bbl@tempc{%
978 \noexpand\newcommand
979 \expandafter\noexpand\csname ##1\endcsname{%
980 \noexpand\protect
981 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
982 \noexpand\newcommand
983 \expandafter\noexpand\csname text##1\endcsname{%
984 \noexpand\foreignlanguage{##2}}}
985 \bbl@tempc}%
986 \bbl@for\bbl@tempa\bbl@tempa{%
987 \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.

```

988 \bbl@trace{Compatibility with language.def}
989 \ifx\directlua\@undefined\else
990 \ifx\bbl@luapatterns\@undefined
991 \input luababel.def
992 \fi
993 \fi
994 \ifx\bbl@languages\@undefined
995 \ifx\directlua\@undefined
996 \openin1 = language.def
997 \ifeof1
998 \closein1
999 \message{I couldn't find the file language.def}
1000 \else
1001 \closein1
1002 \begingroup
1003 \def\addlanguage#1#2#3#4#5{%
1004 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1005 \global\expandafter\let\csname l@#1\endcsname
1006 \csname lang@#1\endcsname
1007 \fi}%
1008 \def\uselanguage#1{%
1009 \input language.def
1010 \endgroup
1011 \fi
1012 \fi

```

```

1013 \chardef\l@english\z@
1014 \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.

```

1015 \def\addto#1#2{%
1016   \ifx#1@undefined
1017     \def#1{#2}%
1018   \else
1019     \ifx#1\relax
1020       \def#1{#2}%
1021     \else
1022       {\toks@expandafter{#1#2}%
1023        \xdef#1{\the\toks@}}%
1024   \fi
1025 \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.

```

1026 \bbl@trace{Hooks}
1027 \newcommand\AddBabelHook[3][]{%
1028   \bbl@iifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{%
1029     \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1030     \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1031     \bbl@iifunset{\bbl@ev@#2@#3@#1}%
1032     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1033     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1034     \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1035 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1036 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1037 \def\bbl@usehooks{\bbl@usehooks@lang\languagename}
1038 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1039   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1040   \def\bbl@elth##1{%
1041     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1042     \bbl@cs{ev@#2@}%
1043     \ifx\languagename\@undefined\else % Test required for Plain (?)
1044       \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1045       \def\bbl@elth##1{%
1046         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1}}%
1047         \bbl@cs{ev@#2@#1}%
1048       \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).

```

1049 \def\bbl@evargs{,% <- don't delete this comma
1050   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1051   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1052   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1053   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1054   beforestart=0,languagename=2,begindocument=1}
1055 \ifx\NewHook\@undefined\else % Test for Plain (?)
1056   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1057   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1058 \fi

```

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

```
1059 \providecommand\PassOptionsToLocale[2]{%
1060   \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.

```
1061 \bbl@trace{Macros for setting language files up}
1062 \def\bbl@ldfinit{%
1063   \let\bbl@screset@empty
1064   \let\BabelStrings\bbl@opt@string
1065   \let\BabelOptions\@empty
1066   \let\BabelLanguages\relax
1067   \ifx\originalTeX\@undefined
1068     \let\originalTeX\@empty
1069   \else
1070     \originalTeX
1071   \fi}
1072 \def\LdfInit#1#2{%
1073   \chardef\atcatcode=\catcode`\@
1074   \catcode`\@=11\relax
1075   \chardef\eqcatcode=\catcode`\=
1076   \catcode`\==12\relax
1077   \ifpackagewith{babel}{ensureinfo=off}}}%
1078   {\ifx\InputIfFileExists\@undefined\else
1079     \bbl@ifunset{bbl@lname@#1}%
1080     {{\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1081       \def\languagenamex{#1}%
1082       \bbl@id@assign
1083       \bbl@load@info{#1}}}%
1084     }%
1085   \fi}%
1086   \expandafter\if\expandafter\@backslashchar
1087     \expandafter\@car\string#2\@nil
1088   \ifx#2\@undefined\else
1089     \ldf@quit{#1}%
1090   \fi
1091 \else
1092   \expandafter\ifx\csname#2\endcsname\relax\else
1093     \ldf@quit{#1}%
1094   \fi
1095 \fi
1096 \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.

```
1097 \def\ldf@quit#1{%
1098   \expandafter\main@language\expandafter{#1}%
1099   \catcode`\@=\atcatcode \let\atcatcode\relax
1100   \catcode`\==\eqcatcode \let\eqcatcode\relax
1101   \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.

```
1102 \def\bbl@afterldf{%
1103   \bbl@afterlang
1104   \let\bbl@afterlang\relax
1105   \let\BabelModifiers\relax
1106   \let\bbl@screset\relax}%
1107 \def\ldf@finish#1{%
1108   \loadlocalcfg{#1}%
1109   \bbl@afterldf
1110   \expandafter\main@language\expandafter{#1}%
1111   \catcode`\@=\atcatcode \let\atcatcode\relax
1112   \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 \LaTeX .

```
1113 \@onlypreamble\LdfInit
1114 \@onlypreamble\ldf@quit
1115 \@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.

```
1116 \def\main@language#1{%
1117   \def\bbl@main@language{#1}%
1118   \let\language\name\bbl@main@language
1119   \let\localename\bbl@main@language
1120   \let\mainlocalename\bbl@main@language
1121   \bbl@id@assign
1122   \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.

```
1123 \def\bbl@beforestart{%
1124   \def\@nolanerr##1{%
1125     \bbl@carg\chardef{l@##1}\z@
1126     \bbl@warning{Undefined language '##1' in aux.\\Reported}}}%
1127   \bbl@usehooks{beforestart}}}%
1128   \global\let\bbl@beforestart\relax}
1129 \AtBeginDocument{%
1130   {\@nameuse\bbl@beforestart}}% Group!
1131   \if@filesw
1132     \providecommand\babel@aux[2]{}%
1133     \immediate\write\@mainaux{unexpanded{%
1134       \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1135     \immediate\write\@mainaux{string\@nameuse\bbl@beforestart}}}%
1136   \fi
1137   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
```

```

1138 \ifbbl@single % must go after the line above.
1139 \renewcommand\selectlanguage[1]{}%
1140 \renewcommand\foreignlanguage[2]{#2}%
1141 \global\let\babel@aux\@gobbletwo % Also as flag
1142 \fi}
1143 %
1144 \ifcase\bbl@engine\or
1145 \AtBeginDocument{\pagedir\bodydir}
1146 \fi

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

1147 \def\select@language@x#1{%
1148 \ifcase\bbl@select@type
1149 \bbl@ifsamestring\language#1\{\}\select@language{#1}}%
1150 \else
1151 \select@language{#1}%
1152 \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.

```

1153 \bbl@trace{Shorhands}
1154 \def\bbl@withactive#1#2{%
1155 \begingroup
1156 \lccode`~=#2\relax
1157 \lowercase{\endgroup#1~}}

```

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

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

```

1158 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1159 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1160 \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1161 \ifx\nfss@catcodes\@undefined\else
1162 \begingroup
1163 \catcode`#1\active
1164 \nfss@catcodes
1165 \ifnum\catcode`#1=\active
1166 \endgroup
1167 \bbl@add\nfss@catcodes{\@makeother#1}%
1168 \else
1169 \endgroup
1170 \fi
1171 \fi}

```

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

For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines " as `\active@prefix " \active@char` (where the first " is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect " or \noexpand "` (i.e., with the original "); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in "safe" contexts (e.g., `\label`), but `\user@active` in

normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char"`.

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

```
1172 \def\bbl@active@def#1#2#3#4{%
1173   \@namedef{#3#1}{%
1174     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1175       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1176     \else
1177       \bbl@afterfi\csname#2@sh@#1\endcsname
1178     \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.

```
1179   \long\@namedef{#3@arg#1}##1{%
1180     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1181       \bbl@afterelse\csname#4#1\endcsname##1%
1182     \else
1183       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1184     \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.

```
1185 \def\initiate@active@char#1{%
1186   \bbl@ifunset{active@char\string#1}%
1187   {\bbl@withactive
1188     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1189   {}}

```

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).

```
1190 \def\@initiate@active@char#1#2#3{%
1191   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1192   \ifx#1\@undefined
1193     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1194   \else
1195     \bbl@csarg\let{oridef@@#2}#1%
1196     \bbl@csarg\edef{oridef@#2}{%
1197       \let\noexpand#1%
1198       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1199   \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`).

```
1200   \ifx#1#3\relax
1201     \expandafter\let\csname normal@char#2\endcsname#3%
1202   \else
1203     \bbl@info{Making #2 an active character}%
1204     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1205     \@namedef{normal@char#2}{%
1206       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1207     \else
1208       \@namedef{normal@char#2}{#3}%
1209     \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).

```

1210 \bbl@restoreactive{#2}%
1211 \AtBeginDocument{%
1212   \catcode`#2\active
1213   \if@filesw
1214     \immediate\write\@mainaux{\catcode`\string#2\active}%
1215   \fi}%
1216 \expandafter\bbl@add@special\csname#2\endcsname
1217 \catcode`#2\active
1218 \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>`).

```

1219 \let\bbl@tempa\@firstoftwo
1220 \if\string^#2%
1221   \def\bbl@tempa{\noexpand\textormath}%
1222 \else
1223   \ifx\bbl@mathnormal\undefined\else
1224     \let\bbl@tempa\bbl@mathnormal
1225   \fi
1226 \fi
1227 \expandafter\edef\csname active@char#2\endcsname{%
1228   \bbl@tempa
1229   {\noexpand\if@safe@actives
1230     \noexpand\expandafter
1231     \expandafter\noexpand\csname normal@char#2\endcsname
1232   \noexpand\else
1233     \noexpand\expandafter
1234     \expandafter\noexpand\csname bbl@doactive#2\endcsname
1235   \noexpand\fi}%
1236   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1237 \bbl@csarg\edef{doactive#2}{%
1238   \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!).

```

1239 \bbl@csarg\edef{active@#2}{%
1240   \noexpand\active@prefix\noexpand#1%
1241   \expandafter\noexpand\csname active@char#2\endcsname}%
1242 \bbl@csarg\edef{normal@#2}{%
1243   \noexpand\active@prefix\noexpand#1%
1244   \expandafter\noexpand\csname normal@char#2\endcsname}%
1245 \bbl@ncarg\let#1\bbl@normal@#2}%

```

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

```

1246 \bbl@active@def#2\user@group{user@active}{language@active}%
1247 \bbl@active@def#2\language@group{language@active}{system@active}%
1248 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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


```

1249 \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1250 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1251 \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1252 {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1253 \if\string'#2%
1254 \let\prim@s\bbl@prim@s
1255 \let\active@math@prime#1%
1256 \fi
1257 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1258 << *More package options >> ≡
1259 \DeclareOption{math=active}{}
1260 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1261 << /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*.

```

1262 \ifpackagewith{babel}{KeepShorthandsActive}%
1263 {\let\bbl@restoreactive\@gobble}%
1264 {\def\bbl@restoreactive#1{%
1265 \bbl@exp{%
1266 \\\AfterBabelLanguage\\CurrentOption
1267 {\catcode`#1=\the\catcode`#1\relax}%
1268 \\\AtEndOfPackage
1269 {\catcode`#1=\the\catcode`#1\relax}}}%
1270 \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.

```

1271 \def\bbl@sh@select#1#2{%
1272 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1273 \bbl@afterelse\bbl@scndcs
1274 \else
1275 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1276 \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.

```

1277 \begingroup
1278 \bbl@ifunset{ifincsname}
1279 {\gdef\active@prefix#1{%
1280 \ifx\protect\@typeset@protect
1281 \else
1282 \ifx\protect\@unexpandable@protect
1283 \noexpand#1%
1284 \else
1285 \protect#1%

```

```

1286     \fi
1287     \expandafter\@gobble
1288     \fi}}
1289 {\gdef\active@prefix#1{%
1290     \ifincsname
1291         \string#1%
1292         \expandafter\@gobble
1293     \else
1294         \ifx\protect\@typeset@protect
1295         \else
1296             \ifx\protect\@unexpandable@protect
1297                 \noexpand#1%
1298             \else
1299                 \protect#1%
1300             \fi
1301         \expandafter\expandafter\expandafter\@gobble
1302     \fi
1303     \fi}}
1304 \endgroup

```

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

```

1305 \newif\if@safe@actives
1306 \@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.

```

1307 \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.

```

1308 \chardef\bbl@activated\z@
1309 \def\bbl@activate#1{%
1310     \chardef\bbl@activated\@ne
1311     \bbl@withactive{\expandafter\let\expandafter}#1%
1312     \csname bbl@active@\string#1\endcsname}
1313 \def\bbl@deactivate#1{%
1314     \chardef\bbl@activated\tw@
1315     \bbl@withactive{\expandafter\let\expandafter}#1%
1316     \csname bbl@normal@\string#1\endcsname}

```

\bbl@firstcs

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

```

1317 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1318 \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.

```

1319 \def\babel@texpdf#1#2#3#4{%
1320   \ifx\texorpdfstring\undefined
1321     \textormath{#1}{#3}%
1322   \else
1323     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1324     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1325   \fi}
1326 %
1327 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1328 \def\@decl@short#1#2#3\@nil#4{%
1329   \def\bbl@tempa{#3}%
1330   \ifx\bbl@tempa\@empty
1331     \expandafter\let\csname #1@sh@\string#2\sel@endcsname\bbl@scndcs
1332     \bbl@ifunset{#1@sh@\string#2@}{}%
1333     {\def\bbl@tempa{#4}%
1334      \expandafter\ifx\csname#1@sh@\string#2@endcsname\bbl@tempa
1335      \else
1336        \bbl@info
1337          {Redefining #1 shorthand \string#2\\%
1338           in language \CurrentOption}%
1339      \fi}%
1340     \@namedef{#1@sh@\string#2@}{#4}%
1341   \else
1342     \expandafter\let\csname #1@sh@\string#2\sel@endcsname\bbl@firstcs
1343     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1344     {\def\bbl@tempa{#4}%
1345      \expandafter\ifx\csname#1@sh@\string#2@\string#3@endcsname\bbl@tempa
1346      \else
1347        \bbl@info
1348          {Redefining #1 shorthand \string#2\string#3\\%
1349           in language \CurrentOption}%
1350      \fi}%
1351     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1352   \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.

```

1353 \def\textormath{%
1354   \ifmmode
1355     \expandafter\@secondoftwo
1356   \else
1357     \expandafter\@firstoftwo
1358   \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’.

```

1359 \def\user@group{user}
1360 \def\language@group{english}
1361 \def\system@group{system}

```

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

```

1362 \def\usesshorthands{%

```

```

1363 \ifstar\bbl@usesesh@s{\bbl@usesesh@x{}}
1364 \def\bbl@usesesh@s#1{%
1365   \bbl@usesesh@x
1366   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1367   {#1}}
1368 \def\bbl@usesesh@x#1#2{%
1369   \bbl@ifshorthand{#2}%
1370   {\def\user@group{user}%
1371    \initiate@active@char{#2}%
1372    #1%
1373    \bbl@activate{#2}}%
1374   {\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.

```

1375 \def\user@language@group{user@\language@group}
1376 \def\bbl@set@user@generic#1#2{%
1377   \bbl@ifunset{user@generic@active#1}%
1378   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1379    \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1380    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1381      \expandafter\noexpand\csname normal@char#1\endcsname}%
1382    \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1383      \expandafter\noexpand\csname user@active#1\endcsname}}%
1384   \@empty}
1385 \newcommand\defineshorthand[3][user]{%
1386   \edef\bbl@tempa{\zap@space#1 \@empty}%
1387   \bbl@for\bbl@tempb\bbl@tempa{%
1388     \if*\expandafter\@car\bbl@tempb\@nil
1389     \edef\bbl@tempb{user\expandafter\@gobble\bbl@tempb}%
1390     \@expandtwoargs
1391     \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1392   }
1393   \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.

```

1394 \def\languageshorthands#1{%
1395   \bbl@ifsamestring{none}{#1}{}%
1396   \bbl@once{short-\localename-#1}{%
1397     \bbl@info{'\localename' activates '#1' shorthands.\@Reported}}%
1398   \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`.

```

1399 \def\aliasshorthand#1#2{%
1400   \bbl@ifshorthand{#2}%
1401   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1402     \ifx\document\@notprerr
1403       \@notshorthand{#2}%
1404     \else
1405       \initiate@active@char{#2}%
1406       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1407       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1408       \bbl@activate{#2}%
1409     \fi

```

```

1410     \fi}%
1411     {\bbl@error{shorthand-is-off}}{\#2}{}}

```

\@notshorthand

```

1412 \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.

```

1413 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1414 \DeclareRobustCommand*\shorthandoff{%
1415   \ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1416 \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.

```

1417 \def\bbl@switch@sh#1#2{%
1418   \ifx#2\@nnil\else
1419     \bbl@ifunset{\bbl@active@\string#2}%
1420     {\bbl@error{not-a-shorthand-b}}{\#2}{}}%
1421     {\ifcase#1%   off, on, off*
1422       \catcode`#2\relax
1423       \or
1424       \catcode`#2\active
1425       \bbl@ifunset{\bbl@shdef@\string#2}%
1426       {}%
1427       {\bbl@withactive{\expandafter\let\expandafter}#2%
1428         \csname bbl@shdef@\string#2\endcsname
1429         \bbl@csarg\let{shdef@\string#2}\relax}%
1430       \ifcase\bbl@activated\or
1431       \bbl@activate{#2}%
1432       \else
1433       \bbl@deactivate{#2}%
1434       \fi
1435       \or
1436       \bbl@ifunset{\bbl@shdef@\string#2}%
1437       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1438       {}%
1439       \csname bbl@oricat@\string#2\endcsname
1440       \csname bbl@oridef@\string#2\endcsname
1441       \fi}%
1442   \bbl@afterfi\bbl@switch@sh#1%
1443   \fi}

```

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

```

1444 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1445 \def\bbl@putsh#1{%
1446   \bbl@ifunset{\bbl@active@\string#1}%
1447   {\bbl@putsh@i#1\@empty\@nnil}%
1448   {\csname bbl@active@\string#1\endcsname}}
1449 \def\bbl@putsh@i#1#2\@nnil{%
1450   \csname\language@group @sh@\string#1@%
1451   \ifx\@empty#2\else\string#2@\fi\endcsname}
1452 %

```

```

1453 \ifx\bbl@opt@shorthands\@nnil\else
1454 \let\bbl@s@initiate@active@char\initiate@active@char
1455 \def\initiate@active@char#1{%
1456   \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1457 \let\bbl@s@switch@sh\bbl@switch@sh
1458 \def\bbl@switch@sh#1#2{%
1459   \ifx#2\@nnil\else
1460     \bbl@afterfi
1461     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1462   \fi}
1463 \let\bbl@s@activate\bbl@activate
1464 \def\bbl@activate#1{%
1465   \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1466 \let\bbl@s@deactivate\bbl@deactivate
1467 \def\bbl@deactivate#1{%
1468   \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1469 \fi

```

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

```

1470 \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.

```

1471 \def\bbl@prim@s{%
1472   \prime\futurelet\@let@token\bbl@pr@m@s}
1473 \def\bbl@if@primes#1#2{%
1474   \ifx#1\@let@token
1475     \expandafter\@firstoftwo
1476   \else\ifx#2\@let@token
1477     \bbl@afterelse\expandafter\@firstoftwo
1478   \else
1479     \bbl@afterfi\expandafter\@secondoftwo
1480   \fi\fi}
1481 \begingroup
1482 \catcode\^=7 \catcode\*= \active \lccode\^=\^
1483 \catcode\'=12 \catcode\"= \active \lccode\"=\^
1484 \lowercase{%
1485   \gdef\bbl@pr@m@s{%
1486     \bbl@if@primes" '%
1487       \pr@@@s
1488       {\bbl@if@primes*\^ \pr@@@t\egroup}}}
1489 \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).

```

1490 \initiate@active@char{~}
1491 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1492 \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.

```

1493 \expandafter\def\csname OT1dqpos\endcsname{127}
1494 \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

```
1495 \ifx\f@encoding\undefined
1496   \def\f@encoding{OT1}
1497 \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.

```
1498 \bbl@trace{Language attributes}
1499 \newcommand\languageattribute[2]{%
1500   \def\bbl@tempc{#1}%
1501   \bbl@fixname\bbl@tempc
1502   \bbl@iflanguage\bbl@tempc{%
1503     \bbl@vforeach{#2}{%
```

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

```
1504     \ifx\bbl@known@attribs\undefined
1505       \in@false
1506     \else
1507       \bbl@xin@{\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1508     \fi
1509     \ifin@
1510       \bbl@warning{%
1511         You have more than once selected the attribute '##1'\%
1512         for language #1. Reported}%
1513     \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.

```
1514       \bbl@info{Activated '##1' attribute for\\%
1515         '\bbl@tempc'. Reported}%
1516       \bbl@exp{%
1517         \\ \bbl@add@list\\ \bbl@known@attribs{\bbl@tempc-##1}}%
1518       \edef\bbl@tempa{\bbl@tempc-##1}%
1519       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1520       {\csname\bbl@tempc @attr##1\endcsname}%
1521       {\@attrerr{\bbl@tempc}{##1}}%
1522     \fi}}
1523 \@onlypreamble\languageattribute
```

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

```
1524 \newcommand*{\@attrerr}[2]{%
1525   \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}`.

```
1526 \def\bbl@declare@ttribute#1#2#3{%
1527   \bbl@xin@{, #2, }{\BabelModifiers,}%
1528   \ifin@
1529     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1530   \fi
1531   \bbl@add@list\bbl@attributes{#1-#2}%
1532   \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.

```

1533 \def\bbl@ifattributeset#1#2#3#4{%
1534   \ifx\bbl@known@attrs\@undefined
1535     \in@false
1536   \else
1537     \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1538   \fi
1539   \ifin@
1540     \bbl@afterelse#3%
1541   \else
1542     \bbl@afterfi#4%
1543   \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.

```

1544 \def\bbl@ifknown@ttrib#1#2{%
1545   \let\bbl@tempa\@secondoftwo
1546   \bbl@loopx\bbl@tempb{#2}{%
1547     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1548   \ifin@
1549     \let\bbl@tempa\@firstoftwo
1550   \else
1551     \fi}%
1552   \bbl@tempa}

```

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

```

1553 \def\bbl@clear@ttribs{%
1554   \ifx\bbl@attributes\@undefined\else
1555     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1556       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1557     \let\bbl@attributes\@undefined
1558   \fi}
1559 \def\bbl@clear@ttrib#1-#2.{%
1560   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1561 \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.

```

1562 \bbl@trace{Macros for saving definitions}
1563 \def\babel@beginsave{\babel@savecnt\z@}

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

1564 \newcount\babel@savecnt
1565 \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.

```
1566 \def\babel@save#1{%
1567   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1568   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1569     \expandafter\expandafter,\bbl@savedextras,}%
1570   \expandafter\in@\bbl@tempa
1571   \ifin@%else
1572     \bbl@add\bbl@savedextras{,{#1,}}%
1573     \bbl@carg\let\babel@number\babel@savecnt#1\relax
1574     \toks@{\expandafter{\originalTeX\let#1=}}%
1575     \bbl@exp{%
1576       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1577     \advance\babel@savecnt@ne
1578   \fi}
1579 \def\babel@savevariable#1{%
1580   \toks@{\expandafter{\originalTeX #1=}}%
1581   \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`.

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

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

```
1587 \def\bbl@redefine@long#1{%
1588   \edef\bbl@tempa{\bbl@stripslash#1}%
1589   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1590   \long\expandafter\def\csname\bbl@tempa\endcsname}
1591 \@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_`.

```
1592 \def\bbl@redefineroobust#1{%
1593   \edef\bbl@tempa{\bbl@stripslash#1}%
1594   \bbl@ifunset{\bbl@tempa\space}%
1595     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1596       \bbl@exp{\def\\#1{\protect<\bbl@tempa\space>}}}%
1597     {\bbl@exp{\let<org@\bbl@tempa><\bbl@tempa\space>}}}%
1598   \@namedef{\bbl@tempa\space}{}
1599 \@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.

```

1600 \def\bbl@frenchspacing{%
1601   \ifnum\the\sfcode`\.\=@m
1602     \let\bbl@nonfrenchspacing\relax
1603   \else
1604     \frenchspacing
1605     \let\bbl@nonfrenchspacing\nonfrenchspacing
1606   \fi}
1607 \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.

```

1608 \let\bbl@elt\relax
1609 \edef\bbl@fs@chars{%
1610   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1611   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1612   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1613 \def\bbl@pre@fs{%
1614   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1615   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1616 \def\bbl@post@fs{%
1617   \bbl@save@sfcodes
1618   \edef\bbl@tempa{\bbl@cl{frspc}}%
1619   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1620   \if u\bbl@tempa      % do nothing
1621   \else\if n\bbl@tempa % non french
1622     \def\bbl@elt##1##2##3{%
1623       \ifnum\sfcode`##1=##2\relax
1624         \babel@savevariable{\sfcode`##1}%
1625         \sfcode`##1=##3\relax
1626       \fi}%
1627     \bbl@fs@chars
1628   \else\if y\bbl@tempa % french
1629     \def\bbl@elt##1##2##3{%
1630       \ifnum\sfcode`##1=##3\relax
1631         \babel@savevariable{\sfcode`##1}%
1632         \sfcode`##1=##2\relax
1633       \fi}%
1634     \bbl@fs@chars
1635   \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.

```

1636 \bbl@trace{Hyphens}
1637 \@onlypreamble\babelhyphenation
1638 \AtEndOfPackage{%
1639   \newcommand\babelhyphenation[2][\@empty]{%
1640     \ifx\bbl@hyphenation@\relax
1641       \let\bbl@hyphenation@\@empty
1642     \fi
1643     \ifx\bbl@hyphlist\@empty\else
1644       \bbl@warning{%
1645         You must not intermingle \string\selectlanguage\space and\\%
1646         \string\babelhyphenation\space or some exceptions will not\\%
1647         be taken into account. Reported}%
1648     \fi

```

```

1649 \ifx\@empty#1%
1650 \protected@edef\bb@hyphenation@\bb@hyphenation\space#2}%
1651 \else
1652 \bb@vforeach{#1}{%
1653 \def\bb@tempa{##1}%
1654 \bb@fixname\bb@tempa
1655 \bb@iflanguage\bb@tempa{%
1656 \bb@csarg\protected@edef\hyphenation@\bb@tempa}{%
1657 \bb@ifunset\bb@hyphenation@\bb@tempa}%
1658 }%
1659 {\csname \bb@hyphenation@\bb@tempa\endcsname\space}%
1660 #2}}}%
1661 \fi}}

```

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

```

1662 \ifx\NewDocumentCommand\@undefined\else
1663 \NewDocumentCommand\babelhyphenmins{sommo}{%
1664 \IfNoValueTF{#2}%
1665 {\protected@edef\bb@hyphenmins@\set@hyphenmins{#3}{#4}}%
1666 \IfValueT{#5}{%
1667 \protected@edef\bb@hyphenatmin@\hyphenationmin=#5\relax}}%
1668 \IfBooleanT{#1}{%
1669 \leftthyphenmin=#3\relax
1670 \rightthyphenmin=#4\relax
1671 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1672 {\edef\bb@tempb{\zap@space#2 \@empty}%
1673 \bb@for\bb@tempa\bb@tempb{%
1674 \@namedef\bb@hyphenmins@\bb@tempa}{\set@hyphenmins{#3}{#4}}%
1675 \IfValueT{#5}{%
1676 \@namedef\bb@hyphenatmin@\bb@tempa}{\hyphenationmin=#5\relax}}}%
1677 \IfBooleanT{#1}{\bb@error{hyphenmins-args}{}}}%
1678 \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.

```

1679 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\zap@space\fi}
1680 \def\bb@t@one{T1}
1681 \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.

```

1682 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1683 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1684 \def\bb@hyphen{%
1685 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
1686 \def\bb@hyphen@i#1#2{%
1687 \lowercase{\bb@ifunset\bb@hy@#1#2\@empty}}%
1688 {\csname \bb@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1689 {\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.

```

1690 \def\bb@usehyphen#1{%
1691 \leavevmode

```

```

1692 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1693 \nobreak\hskip\z@skip}
1694 \def\bbl@usehyphen#1{%
1695 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1696 \def\bbl@hyphenchar{%
1697 \ifnum\hyphenchar\font=\m@ne
1698 \babe\nullhyphen
1699 \else
1700 \char\hyphenchar\font
1701 \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.

```

1702 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1703 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1704 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1705 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1706 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1707 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1708 \def\bbl@hy@repeat{%
1709 \bbl@usehyphen{
1710 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1711 \def\bbl@hy@@repeat{%
1712 \bbl@usehyphen{
1713 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1714 \def\bbl@hy@empty{\hskip\z@skip}
1715 \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.

```

1716 \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.

```

1717 \bbl@trace{Multiencoding strings}
1718 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1719 << *More package options[] ≡
1720 \DeclareOption{nocase}{}
1721 <</More package options[]

```

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

```

1722 << *More package options[] ≡
1723 \let\bbl@opt@strings\@nnil % accept strings=value
1724 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1725 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1726 \def\BabelStringsDefault{generic}
1727 <</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.

```

1728 \@onlypreamble\StartBabelCommands
1729 \def\StartBabelCommands{%
1730   \begingroup
1731   \@tempcnta="7F
1732   \def\bbl@tempa{%
1733     \ifnum\@tempcnta>"FF\else
1734       \catcode\@tempcnta=11
1735       \advance\@tempcnta\@ne
1736       \expandafter\bbl@tempa
1737     \fi}%
1738   \bbl@tempa
1739   <@Macros local to BabelCommands@>
1740   \def\bbl@provstring##1##2{%
1741     \providecommand##1{##2}%
1742     \bbl@tglobal##1}%
1743   \global\let\bbl@scafter\@empty
1744   \let\StartBabelCommands\bbl@startcmds
1745   \ifx\BabelLanguages\relax
1746     \let\BabelLanguages\CurrentOption
1747   \fi
1748   \begingroup
1749   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1750   \StartBabelCommands}
1751 \def\bbl@startcmds{%
1752   \ifx\bbl@screset\@nnil\else
1753     \bbl@usehooks{stopcommands}{}%
1754   \fi
1755   \endgroup
1756   \begingroup
1757   \@ifstar
1758     {\ifx\bbl@opt@strings\@nnil
1759       \let\bbl@opt@strings\BabelStringsDefault
1760       \fi
1761       \bbl@startcmds@i}%
1762     \bbl@startcmds@i}
1763 \def\bbl@startcmds@i##1##2{%
1764   \edef\bbl@L{\zap@space#1 \@empty}%
1765   \edef\bbl@G{\zap@space#2 \@empty}%
1766   \bbl@startcmds@ii}
1767 \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.

```

1768 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1769   \let\SetString\@gobbletwo
1770   \let\bbl@stringdef\@gobbletwo
1771   \let\AfterBabelCommands\@gobble
1772   \ifx\@empty#1%
1773     \def\bbl@sc@label{generic}%
1774     \def\bbl@encstring##1##2{%
1775       \ProvideTextCommandDefault##1{##2}%
1776       \bbl@tglobal##1%
1777       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1778 \let\bbl@sctest\in@true
1779 \else
1780 \let\bbl@sc@charset\space % <- zapped below
1781 \let\bbl@sc@fontenc\space % <- " "
1782 \def\bbl@tempa##1=##2\@nil{%
1783 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1784 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1785 \def\bbl@tempa##1 ##2{% space -> comma
1786 ##1%
1787 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1788 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1789 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1790 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1791 \def\bbl@encstring##1##2{%
1792 \bbl@foreach\bbl@sc@fontenc{%
1793 \bbl@ifunset{T@####1}%
1794 {}%
1795 {\ProvideTextCommand##1{####1}{##2}%
1796 \bbl@tglobal##1%
1797 \expandafter
1798 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1799 \def\bbl@sctest{%
1800 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1801 \fi
1802 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1803 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1804 \let\AfterBabelCommands\bbl@aftercmds
1805 \let\SetString\bbl@setstring
1806 \let\bbl@stringdef\bbl@encstring
1807 \else % i.e., strings=value
1808 \bbl@sctest
1809 \ifin@
1810 \let\AfterBabelCommands\bbl@aftercmds
1811 \let\SetString\bbl@setstring
1812 \let\bbl@stringdef\bbl@provstring
1813 \fi\fi\fi
1814 \bbl@scswitch
1815 \ifx\bbl@G\@empty
1816 \def\SetString##1##2{%
1817 \bbl@error{missing-group}{##1}{}}}%
1818 \fi
1819 \ifx\@empty#1%
1820 \bbl@usehooks{defaultcommands}{}%
1821 \else
1822 \@expandtwoargs
1823 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1824 \fi}

```

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

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

```

1825 \def\bbl@forlang#1#2{%
1826 \bbl@for#1\bbl@L{%
1827 \bbl@xin@{,##1,}{,\BabelLanguages,}%
1828 \ifin@#2\relax\fi}}
1829 \def\bbl@scswitch{%
1830 \bbl@forlang\bbl@tempa{%
1831 \ifx\bbl@G\@empty\else

```

```

1832     \ifx\SetString@gobbletwo\else
1833     \edef\bbl@GL{\bbl@G\bbl@tempa}%
1834     \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1835     \ifin@else
1836     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1837     \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1838     \fi
1839     \fi
1840 \fi}}
1841 \AtEndOfPackage{%
1842   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1843   \let\bbl@scswitch\relax}
1844 \@onlypreamble\EndBabelCommands
1845 \def\EndBabelCommands{%
1846   \bbl@usehooks{stopcommands}{}}%
1847 \endgroup
1848 \endgroup
1849 \bbl@scafter}
1850 \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.

```

1851 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1852   \bbl@forlang\bbl@tempa{%
1853     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1854     \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1855     {\bbl@exp{%
1856       \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1857     }%
1858     \def\BabelString{#2}%
1859     \bbl@usehooks{stringprocess}{}}%
1860     \expandafter\bbl@stringdef
1861     \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.

```

1862 \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.

```

1863 << *Macros local to BabelCommands >> ≡
1864 \def\SetStringLoop##1##2{%
1865   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1866   \count@\z@
1867   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1868     \advance\count@\@ne
1869     \toks@\expandafter{\bbl@tempa}%
1870     \bbl@exp{%
1871       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1872       \count@=\the\count@\relax}}}%
1873 << /Macros local to BabelCommands >>

```

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

```

1874 \def\bbl@aftercmds#1{%
1875   \toks@\expandafter{\bbl@scafter#1}%
1876   \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.

```

1877 <<*Macros local to BabelCommands>> ≡
1878   \newcommand\SetCase[3][]{%
1879     \def\bbl@tempa####1####2{%
1880       \ifx####1\@empty\else
1881         \bbl@carg\bbl@add{extras\CurrentOption}{%
1882           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1883           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1884           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1885           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1886       \expandafter\bbl@tempa
1887     \fi}%
1888   \bbl@tempa##1\@empty\@empty
1889   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1890 <</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.

```

1891 <<*Macros local to BabelCommands>> ≡
1892   \newcommand\SetHyphenMap[1]{%
1893     \bbl@forlang\bbl@tempa{%
1894       \expandafter\bbl@stringdef
1895       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1896 <</Macros local to BabelCommands>>

```

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

```

1897 \newcommand\BabelLower[2]{% one to one.
1898   \ifnum\lccode#1=#2\else
1899     \babel@savevariable{\lccode#1}%
1900     \lccode#1=#2\relax
1901   \fi}
1902 \newcommand\BabelLowerMM[4]{% many-to-many
1903   \@tempcnta=#1\relax
1904   \@tempcntb=#4\relax
1905   \def\bbl@tempa{%
1906     \ifnum\@tempcnta>#2\else
1907       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1908       \advance\@tempcnta#3\relax
1909       \advance\@tempcntb#3\relax
1910       \expandafter\bbl@tempa
1911     \fi}%
1912   \bbl@tempa}
1913 \newcommand\BabelLowerM0[4]{% many-to-one
1914   \@tempcnta=#1\relax
1915   \def\bbl@tempa{%
1916     \ifnum\@tempcnta>#2\else
1917       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1918       \advance\@tempcnta#3
1919       \expandafter\bbl@tempa
1920     \fi}%
1921   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1922 <<*More package options>> ≡
1923 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1924 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1925 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1926 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1927 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1928 <</More package options>>

```


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

```

1929 \AtEndOfPackage{%
1930   \ifx\bbbl@opt@hyphenmap\@undefined
1931     \bbbl@xin@{,}\bbbl@language@opts}%
1932     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1933   \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.

```

1934 \newcommand\setlocalecaption{%
1935   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1936 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1937   \bbbl@trim@def\bbbl@tempa{#2}%
1938   \bbbl@xin@{.template}\bbbl@tempa}%
1939   \ifin@
1940     \bbbl@ini@captions@template{#3}{#1}%
1941   \else
1942     \edef\bbbl@tempd{%
1943       \expandafter\expandafter\expandafter
1944       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1945     \bbbl@xin@
1946       {\expandafter\string\csname #2name\endcsname}%
1947       {\bbbl@tempd}%
1948     \ifin@ % Renew caption
1949       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
1950     \ifin@
1951       \bbbl@exp{%
1952         \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1953         {\bbbl@scset\<#2name>\<#1#2name>}%
1954         {}}%
1955       \else % Old way converts to new way
1956         \bbbl@ifunset{#1#2name}%
1957         {\bbbl@exp{%
1958           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1959           \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1960           {\def\<#2name>\<#1#2name>}}%
1961           {}}%
1962         {}}%
1963     \fi
1964   \else
1965     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
1966     \ifin@ % New way
1967       \bbbl@exp{%
1968         \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
1969         \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1970         {\bbbl@scset\<#2name>\<#1#2name>}%
1971         {}}%
1972       \else % Old way, but defined in the new way
1973         \bbbl@exp{%
1974           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1975           \\bbbl@ifsamestring{\bbbl@tempa}\language}%
1976           {\def\<#2name>\<#1#2name>}}%
1977           {}}%
1978     \fi%
1979   \fi
1980   \@namedef{#1#2name}{#3}%
1981   \toks@ \expandafter\bbbl@captionslist}%
1982   \bbbl@exp{\in@{\<#2name>}\the\toks@}%
1983   \ifin@ \else
1984     \bbbl@exp{\bbbl@add\bbbl@captionslist{\<#2name>}}%

```

```

1985      \bbl@tglobal\bbl@captionslist
1986      \fi
1987      \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.

```

1988 \bbl@trace{Macros related to glyphs}
1989 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1990      \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
1991      \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.

```

1992 \def\save@sf@q#1{\leavevmode
1993      \begingroup
1994      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1995      \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.

```

1996 \ProvideTextCommand{\quotedblbase}{OT1}{%
1997      \save@sf@q{\set@low@box{\textquotedblright\}}%
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{\quotedblbase}{%
2000      \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2001 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2002      \save@sf@q{\set@low@box{\textquoteright\}}%
2003      \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.

```

2004 \ProvideTextCommandDefault{\quotesinglbase}{%
2005      \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

2006 \ProvideTextCommand{\guillemetleft}{OT1}{%
2007      \ifmmode
2008          \ll
2009      \else
2010          \save@sf@q{\nobreak
2011              \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2012          \fi}
2013 \ProvideTextCommand{\guillemetright}{OT1}{%
2014      \ifmmode
2015          \gg
2016      \else
2017          \save@sf@q{\nobreak
2018              \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%

```

```

2019 \fi}
2020 \ProvideTextCommand{\guillemotleft}{OT1}{%
2021 \ifmmode
2022 \ll
2023 \else
2024 \save@sf@q{\nobreak
2025 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2026 \fi}
2027 \ProvideTextCommand{\guillemotright}{OT1}{%
2028 \ifmmode
2029 \gg
2030 \else
2031 \save@sf@q{\nobreak
2032 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2033 \fi}

```

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

```

2034 \ProvideTextCommandDefault{\guillemetleft}{%
2035 \UseTextSymbol{OT1}{\guillemetleft}}
2036 \ProvideTextCommandDefault{\guillemetright}{%
2037 \UseTextSymbol{OT1}{\guillemetright}}
2038 \ProvideTextCommandDefault{\guillemotleft}{%
2039 \UseTextSymbol{OT1}{\guillemotleft}}
2040 \ProvideTextCommandDefault{\guillemotright}{%
2041 \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2042 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2043 \ifmmode
2044 <%
2045 \else
2046 \save@sf@q{\nobreak
2047 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2048 \fi}
2049 \ProvideTextCommand{\guilsinglright}{OT1}{%
2050 \ifmmode
2051 >%
2052 \else
2053 \save@sf@q{\nobreak
2054 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2055 \fi}

```

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

```

2056 \ProvideTextCommandDefault{\guilsinglleft}{%
2057 \UseTextSymbol{OT1}{\guilsinglleft}}
2058 \ProvideTextCommandDefault{\guilsinglright}{%
2059 \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.

```

2060 \DeclareTextCommand{\ij}{OT1}{%
2061 i\kern-0.02em\bbl@allowhyphens j}
2062 \DeclareTextCommand{\IJ}{OT1}{%
2063 I\kern-0.02em\bbl@allowhyphens J}
2064 \DeclareTextCommand{\ij}{T1}{\char188}
2065 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2066 \ProvideTextCommandDefault{\ij}{%
2067   \UseTextSymbol{OT1}{\ij}}
2068 \ProvideTextCommandDefault{\IJ}{%
2069   \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).

```
2070 \def\crrtic@{\hrule height0.1ex width0.3em}
2071 \def\crttic@{\hrule height0.1ex width0.33em}
2072 \def\ddj@{%
2073   \setbox0\hbox{d}\dimen@=\ht0
2074   \advance\dimen@lex
2075   \dimen@.45\dimen@
2076   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2077   \advance\dimen@ii.5ex
2078   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2079 \def\DDJ@{%
2080   \setbox0\hbox{D}\dimen@=.55\ht0
2081   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2082   \advance\dimen@ii.15ex % correction for the dash position
2083   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2084   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2085   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2086 %
2087 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2088 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2089 \ProvideTextCommandDefault{\dj}{%
2090   \UseTextSymbol{OT1}{\dj}}
2091 \ProvideTextCommandDefault{\DJ}{%
2092   \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.

```
2093 \DeclareTextCommand{\SS}{OT1}{SS}
2094 \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.

```
2095 \ProvideTextCommandDefault{\glq}{%
2096   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2097 \ProvideTextCommand{\grq}{T1}{%
2098   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2099 \ProvideTextCommand{\grq}{TU}{%
2100   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2101 \ProvideTextCommand{\grq}{OT1}{%
2102   \save@sf@q{\kern-.0125em
2103     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2104   }
```

```

2104 \kern.07em\relax}}
2105 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2106 \ProvideTextCommandDefault{\glqq}{%
2107 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2108 \ProvideTextCommand{\grqq}{T1}{%
2109 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2110 \ProvideTextCommand{\grqq}{TU}{%
2111 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2112 \ProvideTextCommand{\grqq}{0T1}{%
2113 \save@sf@q{\kern-.07em
2114 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2115 \kern.07em\relax}}
2116 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2117 \ProvideTextCommandDefault{\flq}{%
2118 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2119 \ProvideTextCommandDefault{\frq}{%
2120 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```

2121 \ProvideTextCommandDefault{\flqq}{%
2122 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2123 \ProvideTextCommandDefault{\frqq}{%
2124 \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).

```

2125 \def\umlauthigh{%
2126 \def\bbl@umlauta##1{\leavevmode\bgroup%
2127 \accent\csname\f@encoding dqpos\endcsname
2128 ##1\bbl@allowhyphens\egroup}%
2129 \let\bbl@umlaute\bbl@umlauta}
2130 \def\umlautlow{%
2131 \def\bbl@umlauta{\protect\lower@umlaut}}
2132 \def\umlautelow{%
2133 \def\bbl@umlaute{\protect\lower@umlaut}}
2134 \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.

```
2135 \expandafter\ifx\csname U@D\endcsname\relax
2136   \csname newdimen\endcsname\U@D
2137 \fi
```

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

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

```
2138 \def\lower@umlaut#1{%
2139   \leavevmode\bgroup
2140     \U@D lex%
2141     {\setbox\z@\hbox{%
2142       \char\csname f@encoding dqpos\endcsname}%
2143       \dimen@ -.45ex\advance\dimen@ \ht\z@
2144       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2145     \accent\csname f@encoding dqpos\endcsname
2146     \fontdimen5\font\U@D #1%
2147   \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).

```
2148 \AtBeginDocument{%
2149   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2150   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2151   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2152   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
2153   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2154   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2155   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2156   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2157   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2158   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2159   \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`.

```
2160 \ifx\l@english\@undefined
2161   \chardef\l@english\z@
2162 \fi
2163 % The following is used to cancel rules in ini files (see Amharic).
2164 \ifx\l@unhyphenated\@undefined
2165   \newlanguage\l@unhyphenated
2166 \fi
```

4.16. Layout

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

```
2167 \bbl@trace{Bidi layout}
2168 \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.

```
2169 \bbl@trace{Input engine specific macros}
2170 \ifcase\bbl@engine
2171   \input txtbabel.def
2172 \or
2173   \input luababel.def
2174 \or
2175   \input xebabel.def
2176 \fi
2177 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2178 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2179 \ifx\babelposthyphenation\undefined
2180   \let\babelposthyphenation\babelprehyphenation
2181   \let\babelpatterns\babelprehyphenation
2182   \let\babelcharproperty\babelprehyphenation
2183 \fi
2184 /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.

```
2185 {*package
2186 \bbl@trace{Creating languages and reading ini files}
2187 \let\bbl@extend@ini@gobble
2188 \newcommand\babelprovide[2][]{%
2189   \let\bbl@savelangname\languagename
2190   \edef\bbl@savelocaleid{\the\localeid}%
2191   % Set name and locale id
2192   \edef\languagename{#2}%
2193   \bbl@id@assign
2194   % Initialize keys
2195   \bbl@vforeach{captions,date,import,main,script,language,%
2196     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2197     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2198     Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2199     @import}%
2200     {\bbl@csarg\let{KVP@##1}\@nnil}%
2201   \global\let\bbl@release@transforms@empty
2202   \global\let\bbl@release@casing@empty
2203   \let\bbl@calendars@empty
2204   \global\let\bbl@inidata@empty
2205   \global\let\bbl@extend@ini@gobble
2206   \global\let\bbl@included@inis@empty
2207   \gdef\bbl@key@list{;}%
2208   \bbl@ifunset\bbl@passto@#2{%
2209     {\def\bbl@tempa{#1}}%
2210     {\bbl@exp{\def\\bbl@tempa{[\bbl@passto@#2],\unexpanded{#1}}}%
2211     \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2212       \in@{/}{##1}% With /, (re)sets a value in the ini
2213       \ifin@
2214         \bbl@renewinikey##1\@{##2}%
2215       \else
2216         \bbl@csarg\ifx{KVP@##1}\@nnil\else
2217           \bbl@error{unknown-provide-key}{##1}{}}{}%
2218       \fi
2219       \bbl@csarg\def{KVP@##1}{##2}%
2220     \fi}%
2221 }
```

```

2221 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2222 \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw}%
2223 % == init ==
2224 \ifx\bbl@screset\@undefined
2225 \bbl@ldfinit
2226 \fi
2227 % ==
2228 % If there is no import (last wins), use @import (internal, there
2229 % must be just one). To consider any order (because
2230 % \PassOptionsToLocale).
2231 \ifx\bbl@KVP@import\@nnil
2232 \let\bbl@KVP@import\bbl@KVP@@import
2233 \fi
2234 % == date (as option) ==
2235 % \ifx\bbl@KVP@date\@nnil\else
2236 % \fi
2237 % ==
2238 \let\bbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2239 \ifcase\bbl@howloaded
2240 \let\bbl@lbfkflag\@empty % new
2241 \else
2242 \ifx\bbl@KVP@hyphenrules\@nnil\else
2243 \let\bbl@lbfkflag\@empty
2244 \fi
2245 \ifx\bbl@KVP@import\@nnil\else
2246 \let\bbl@lbfkflag\@empty
2247 \fi
2248 \fi
2249 % == import, captions ==
2250 \ifx\bbl@KVP@import\@nnil\else
2251 \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
2252 {\ifx\bbl@initoload\relax
2253 \begingroup
2254 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2255 \bbl@input@texini{#2}%
2256 \endgroup
2257 \else
2258 \xdef\bbl@KVP@import{\bbl@initoload}%
2259 \fi}%
2260 {}%
2261 \let\bbl@KVP@date\@empty
2262 \fi
2263 \let\bbl@KVP@captions@@\bbl@KVP@captions
2264 \ifx\bbl@KVP@captions\@nnil
2265 \let\bbl@KVP@captions\bbl@KVP@import
2266 \fi
2267 % ==
2268 \ifx\bbl@KVP@transforms\@nnil\else
2269 \bbl@replace\bbl@KVP@transforms{ }{,}%
2270 \fi
2271 % ==
2272 \ifx\bbl@KVP@mapdot\@nnil\else
2273 \def\bbl@tempa{\@empty}%
2274 \ifx\bbl@KVP@mapdot\bbl@tempa\else
2275 \bbl@exp{\gdef\<bbl@map@@.\@language>{\[bbl@KVP@mapdot]}}%
2276 \fi
2277 \fi
2278 % Load ini
2279 % -----
2280 \ifcase\bbl@howloaded
2281 \bbl@provide@new{#2}%
2282 \else
2283 \bbl@ifblank{#1}%

```



```

2284     {}% With \bbl@load@basic below
2285     {\bbl@provide@renew{#2}}%
2286 \fi
2287 % Post tasks
2288 % -----
2289 % == subsequent calls after the first provide for a locale ==
2290 \ifx\bbl@inidata\@empty\else
2291     \bbl@extend@ini{#2}%
2292 \fi
2293 % == ensure captions ==
2294 \ifx\bbl@KVP@captions\@nnil\else
2295     \bbl@ifunset{bbl@extracaps@#2}%
2296         {\bbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2297         {\bbl@exp{\\babelensure[exclude=\\today,
2298             include=\[bbl@extracaps@#2]]{#2}}}%
2299     \bbl@ifunset{bbl@ensure@\language}%
2300         {\bbl@exp{%
2301             \\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2302                 \\foreignlanguage{\language}%
2303                 {###1}}}%
2304         }%
2305     \bbl@exp{%
2306         \\bbl@tglobal\<bbl@ensure@\language>%
2307         \\bbl@tglobal\<bbl@ensure@\language\space>}%
2308 \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.

```

2309 \bbl@load@basic{#2}%
2310 % == script, language ==
2311 % Override the values from ini or defines them
2312 \ifx\bbl@KVP@script\@nnil\else
2313     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2314 \fi
2315 \ifx\bbl@KVP@language\@nnil\else
2316     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2317 \fi
2318 \ifcase\bbl@engine\or
2319     \bbl@ifunset{bbl@chrng@\language}{}%
2320     {\directlua{
2321         Babel.set_chrngs_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2322 \fi
2323 % == Line breaking: intraspace, intrapenalty ==
2324 % For CJK, East Asian, Southeast Asian, if interspace in ini
2325 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2326     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2327 \fi
2328 \bbl@provide@intraspace
2329 % == Line breaking: justification ==
2330 \ifx\bbl@KVP@justification\@nnil\else
2331     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2332 \fi
2333 \ifx\bbl@KVP@linebreaking\@nnil\else
2334     \bbl@xin@{,\bbl@KVP@linebreaking,%
2335         {,elongated,kashida,cjk,padding,unhyphenated},%
2336     \ifin@
2337         \bbl@csarg\xdef
2338             {lnbrk@\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2339     \fi
2340 \fi
2341 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2342 \ifin@e\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi

```

```

2343 \ifin@bbl@arabicjust\fi
2344 \bbl@xin@{/p}{/\bbl@cl\lnbrk}}%
2345 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2346 % == Line breaking: hyphenate.other.(locale|script) ==
2347 \ifx\bbl@lbfkflag\empty
2348   \bbl@ifunset{bbl@hyotl@\language\language}%
2349   {\bbl@csarg\bbl@replace{hyotl@\language\language}{ }{,}%
2350     \bbl@startcommands*\language\language}%
2351     \bbl@csarg\bbl@foreach{hyotl@\language\language}%
2352     \ifcase\bbl@engine
2353     \ifnum##1<257
2354       \SetHyphenMap{\BabelLower{##1}{##1}}%
2355     \fi
2356     \else
2357       \SetHyphenMap{\BabelLower{##1}{##1}}%
2358     \fi}%
2359   \bbl@endcommands}%
2360 \bbl@ifunset{bbl@hyots@\language\language}%
2361 {\bbl@csarg\bbl@replace{hyots@\language\language}{ }{,}%
2362   \bbl@csarg\bbl@foreach{hyots@\language\language}%
2363   \ifcase\bbl@engine
2364   \ifnum##1<257
2365     \global\lccode##1=##1\relax
2366   \fi
2367   \else
2368     \global\lccode##1=##1\relax
2369   \fi}}%
2370 \fi
2371 % == Counters: maparabic ==
2372 % Native digits, if provided in ini (TeX level, xe and lua)
2373 \ifcase\bbl@engine\else
2374   \bbl@ifunset{bbl@dgnat@\language\language}%
2375   {\expandafter\ifx\csname bbl@dgnat@\language\language\endcsname\empty\else
2376     \expandafter\expandafter\expandafter
2377     \bbl@setdigits\csname bbl@dgnat@\language\language\endcsname
2378     \ifx\bbl@KVP@maparabic\@nnil\else
2379       \ifx\bbl@latinarabic\@undefined
2380         \expandafter\let\expandafter\@arabic
2381         \csname bbl@counter@\language\language\endcsname
2382       \else % i.e., if layout=counters, which redefines \@arabic
2383         \expandafter\let\expandafter\bbl@latinarabic
2384         \csname bbl@counter@\language\language\endcsname
2385       \fi
2386     \fi
2387   \fi}%
2388 \fi
2389 % == Counters: mapdigits ==
2390 % > luababel.def
2391 % == Counters: alph, Alph ==
2392 \ifx\bbl@KVP@alph\@nnil\else
2393   \bbl@exp{%
2394     \\bbl@add<bbl@preextras@\language\language>{%
2395       \\bbl@save\\@alph
2396       \let\\@alph<bbl@cntr@bbl@KVP@alph @\language\language>}}%
2397 \fi
2398 \ifx\bbl@KVP@Alph\@nnil\else
2399   \bbl@exp{%
2400     \\bbl@add<bbl@preextras@\language\language>{%
2401       \\bbl@save\\@Alph
2402       \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language\language>}}%
2403 \fi
2404 % == Counters: mapdot ==
2405 \ifx\bbl@KVP@mapdot\@nnil\else

```

```

2406 \bbl@foreach\bbl@list@the{%
2407 \bbl@ifunset{the##1}}{%
2408 {{\bbl@ncarg\let\bbl@tempd{the##1}%
2409 \bbl@carg\bbl@sreplace{the##1}{.}{\bbl@map@lbl{.}}%
2410 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2411 \bbl@exp{\gdef<the##1>{{\the##1}}}%
2412 \fi}}%
2413 \edef\bbl@tempb{enumi,enumii,enumiii,enumiv}%
2414 \bbl@foreach\bbl@tempb{%
2415 \bbl@ifunset{label##1}}{%
2416 {{\bbl@ncarg\let\bbl@tempd{label##1}%
2417 \bbl@carg\bbl@sreplace{label##1}{.}{\bbl@map@lbl{.}}%
2418 \expandafter\ifx\csname label##1\endcsname\bbl@tempd\else
2419 \bbl@exp{\gdef<label##1>{{\label##1}}}%
2420 \fi}}%
2421 \fi
2422 % == Casing ==
2423 \bbl@release@casing
2424 \ifx\bbl@KVP@casing\@nnil\else
2425 \bbl@csarg\xdef{casing@}\languagename}%
2426 {\@nameuse{\bbl@casing@}\languagename}\bbl@maybextx\bbl@KVP@casing}%
2427 \fi
2428 % == Calendars ==
2429 \ifx\bbl@KVP@calendar\@nnil
2430 \edef\bbl@KVP@calendar{\bbl@ccl{calpr}}%
2431 \fi
2432 \def\bbl@tempe##1 ##2\@@{% Get first calendar
2433 \def\bbl@tempa{##1}}%
2434 \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@@}%
2435 \def\bbl@tempe##1.##2.##3\@@{%
2436 \def\bbl@tempc{##1}%
2437 \def\bbl@tempb{##2}}%
2438 \expandafter\bbl@tempe\bbl@tempa..\@@
2439 \bbl@csarg\edef{calpr@}\languagename}{%
2440 \ifx\bbl@tempc\@empty\else
2441 calendar=\bbl@tempc
2442 \fi
2443 \ifx\bbl@tempb\@empty\else
2444 ,variant=\bbl@tempb
2445 \fi}%
2446 % == engine specific extensions ==
2447 % Defined in XXXbabel.def
2448 \bbl@provide@extra{#2}%
2449 % == require.babel in ini ==
2450 % To load or reload the babel-*.tex, if require.babel in ini
2451 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2452 \bbl@ifunset{\bbl@rqtex@}\languagename}{%
2453 {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2454 \let\BabelBeforeIni\@gobbletwo
2455 \chardef\atcatcode=\catcode\@
2456 \catcode\@=11\relax
2457 \def\CurrentOption{#2}%
2458 \bbl@input@texini{\bbl@cs{rqtex@}\languagename}}%
2459 \catcode\@=\atcatcode
2460 \let\atcatcode\relax
2461 \global\bbl@csarg\let{rqtex@}\languagename}\relax
2462 \fi}%
2463 \bbl@foreach\bbl@calendars{%
2464 \bbl@ifunset{\bbl@ca@##1}}{%
2465 \chardef\atcatcode=\catcode\@
2466 \catcode\@=11\relax
2467 \InputIfFileExists{babel-ca-##1.tex}{}}%
2468 \catcode\@=\atcatcode

```

```

2469     \let\atcatcode\relax}%
2470   {}}%
2471 \fi
2472 % == frenchspacing ==
2473 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2474 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2475 \ifin@
2476   \bbbl@extras@wrap{\bbbl@pre@fs}%
2477   {\bbbl@pre@fs}%
2478   {\bbbl@post@fs}%
2479 \fi
2480 % == transforms ==
2481 % > luababel.def
2482 \def\CurrentOption{#2}%
2483 \@nameuse{\bbbl@icsave@#2}%
2484 % == main ==
2485 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2486   \let\language\bbbl@savelangname
2487   \chardef\localeid\bbbl@savelocaleid\relax
2488 \fi
2489 % == hyphenrules (apply if current) ==
2490 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2491   \ifnum\bbbl@savelocaleid=\localeid
2492     \language\@nameuse{\bbbl@language}%
2493   \fi
2494 \fi}

```

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

```

2495 \def\bbbl@provide@new#1{%
2496   \@namedef{date#1}{} marks lang exists - required by \StartBabelCommands
2497   \@namedef{extras#1}{}
2498   \@namedef{noextras#1}{}
2499   \bbbl@startcommands*{#1}{captions}%
2500   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2501     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2502       \ifx##1\@nnil\else
2503         \bbbl@exp{%
2504           \\SetString\\##1{%
2505             \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}%
2506           \expandafter\bbbl@tempb
2507         \fi}%
2508     \expandafter\bbbl@tempb\bbbl@captionslist\@nnil
2509   \else
2510     \ifx\bbbl@initoload\relax
2511       \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2512     \else
2513       \bbbl@read@ini{\bbbl@initoload}2% % Same
2514     \fi
2515   \fi
2516   \StartBabelCommands*{#1}{date}%
2517   \ifx\bbbl@KVP@date\@nnil
2518     \bbbl@exp{%
2519       \\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2520   \else
2521     \bbbl@savetoday
2522     \bbbl@savedate
2523   \fi
2524   \bbbl@endcommands
2525   \bbbl@load@basic{#1}%
2526   % == hyphenmins == (only if new)
2527   \bbbl@exp{%
2528     \gdef\<#1hyphenmins>{%

```

```

2529      {\bbl@ifunset{\bbl@lftm@#1}{2}{\bbl@cs{lftm@#1}}}%
2530      {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
2531 % == hyphenrules (also in renew) ==
2532 \bbl@provide@hyphens{#1}%
2533 % == main ==
2534 \ifx\bbl@KVP@main\@nnil\else
2535   \expandafter\main@language\expandafter{#1}%
2536 \fi}
2537 %
2538 \def\bbl@provide@renew#1{%
2539   \ifx\bbl@KVP@captions\@nnil\else
2540     \StartBabelCommands*{#1}{captions}%
2541     \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2542   \EndBabelCommands
2543 \fi
2544 \ifx\bbl@KVP@date\@nnil\else
2545   \StartBabelCommands*{#1}{date}%
2546   \bbl@savetoday
2547   \bbl@savestate
2548   \EndBabelCommands
2549 \fi
2550 % == hyphenrules (also in new) ==
2551 \ifx\bbl@lbkflag\@empty
2552   \bbl@provide@hyphens{#1}%
2553 \fi
2554 % == main ==
2555 \ifx\bbl@KVP@main\@nnil\else
2556   \expandafter\main@language\expandafter{#1}%
2557 \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.

```

2558 \def\bbl@load@basic#1{%
2559   \ifcase\bbl@howloaded\or\or
2560     \ifcase\csname bbl@llevel@\language\endcsname
2561       \bbl@csarg\let\lname@\language\relax
2562     \fi
2563 \fi
2564 \bbl@ifunset{\bbl@lname@#1}%
2565   {\def\BabelBeforeIni##1##2{%
2566     \begin{group}
2567       \let\bbl@ini@captions@aux\@gobbletwo
2568       \def\bbl@inidate####1.####2.####3.####4\relax####5####6}%
2569       \bbl@read@ini{##1}1%
2570       \ifx\bbl@initoload\relax\endinput\fi
2571     \endgroup}%
2572   \begin{group} % boxed, to avoid extra spaces:
2573     \ifx\bbl@initoload\relax
2574       \bbl@input@texini{#1}%
2575     \else
2576       \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2577     \fi
2578   \endgroup}%
2579   {}}

```

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).

```

2580 \def\bbl@load@info#1{%
2581   \def\BabelBeforeIni##1##2{%
2582     \begin{group}
2583       \bbl@read@ini{##1}0%

```

```

2584 \endinput % babel- .tex may contain onlypreamble's
2585 \endgroup}% boxed, to avoid extra spaces:
2586 {\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.

```

2587 \def\bbl@provide@hyphens#1{%
2588 \@tempcnta\m@ne % a flag
2589 \ifx\bbl@KVP@hyphenrules\@nnil\else
2590 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2591 \bbl@foreach\bbl@KVP@hyphenrules{%
2592 \ifnum\@tempcnta=\m@ne % if not yet found
2593 \bbl@ifsamestring{##1}{+}%
2594 {\bbl@carg\addlanguage{l@#1}}%
2595 }%
2596 \bbl@ifunset{l@#1}% After a possible +
2597 }%
2598 {\@tempcnta\@nameuse{l@#1}}%
2599 \fi}%
2600 \ifnum\@tempcnta=\m@ne
2601 \bbl@warning{%
2602 Requested 'hyphenrules' for '\language' not found:\\%
2603 \bbl@KVP@hyphenrules.\\%
2604 Using the default value. Reported}%
2605 \fi
2606 \fi
2607 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2608 \ifx\bbl@KVP@captions@\@nnil
2609 \bbl@ifunset\bbl@hyphr{#1}{}% use value in ini, if exists
2610 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2611 }%
2612 {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2613 }% if hyphenrules found:
2614 {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}%
2615 \fi
2616 \fi
2617 \bbl@ifunset{l@#1}%
2618 {\ifnum\@tempcnta=\m@ne
2619 \bbl@carg\adddialect{l@#1}\language
2620 \else
2621 \bbl@carg\adddialect{l@#1}\@tempcnta
2622 \fi}%
2623 {\ifnum\@tempcnta=\m@ne\else
2624 \global\bbl@carg\chardef{l@#1}\@tempcnta
2625 \fi}}

```

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

```

2626 \def\bbl@input@texini#1{%
2627 \bbl@bsphack
2628 \bbl@exp{%
2629 \catcode`\\%=14 \catcode`\\%=0
2630 \catcode`\\%=1 \catcode`\\%=2
2631 \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2632 \catcode`\\%=the\catcode`%\relax
2633 \catcode`\\%=the\catcode`\\relax
2634 \catcode`\\%=the\catcode`\\relax
2635 \catcode`\\%=the\catcode`\\relax}%
2636 \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.

```

2637 \def\bbl@iniline#1\bbl@iniline{%

```

```

2638 \ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2639 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2640 \def\bbl@iniskip#1\@@{\% if starts with ;
2641 \def\bbl@inistore#1=#2\@@{\% full (default)
2642 \bbl@trim\def\bbl@tempa{#1}%
2643 \bbl@trim\toks@{#2}%
2644 \bbl@ifsamestring{\bbl@tempa}{\@include}%
2645 {\bbl@read@subini{\the\toks@}}%
2646 {\bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2647 \ifin@
2648 \bbl@xin@{,identification/include.}%
2649 {,\bbl@section/\bbl@tempa}%
2650 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2651 \bbl@exp{%
2652 \\\g@addto@macro\\bbl@inidata{%
2653 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2654 \fi}}
2655 \def\bbl@inistore@min#1=#2\@@{\% minimal (maybe set in \bbl@read@ini)
2656 \bbl@trim\def\bbl@tempa{#1}%
2657 \bbl@trim\toks@{#2}%
2658 \bbl@xin@{.identification.}{.\bbl@section.}%
2659 \ifin@
2660 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2661 \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2662 \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.

```

2663 \def\bbl@loop@ini#1{%
2664 \loop
2665 \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2666 \endlinechar\m@ne
2667 \read#1 to \bbl@line
2668 \endlinechar\^^M
2669 \ifx\bbl@line\@empty\else
2670 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2671 \fi
2672 \repeat}
2673 %
2674 \def\bbl@read@subini#1{%
2675 \ifx\bbl@readsubstream\undefined
2676 \csname newread\endcsname\bbl@readsubstream
2677 \fi
2678 \openin\bbl@readsubstream=babel-#1.ini
2679 \ifeof\bbl@readsubstream
2680 \bbl@error{no-ini-file}{#1}{}%
2681 \else
2682 {\bbl@loop@ini\bbl@readsubstream}%
2683 \fi
2684 \closein\bbl@readsubstream}
2685 %

```

```

2686 \ifx\bbl@readstream\@undefined
2687 \csname newread\endcsname\bbl@readstream
2688 \fi
2689 \def\bbl@read@ini#1#2{%
2690 \global\let\bbl@extend@ini\@gobble
2691 \openin\bbl@readstream=babel-#1.ini
2692 \ifeof\bbl@readstream
2693 \bbl@error{no-ini-file}{#1}{}{}%
2694 \else
2695 % == Store ini data in \bbl@inidata ==
2696 \catcode\ =10 \catcode`\"=12
2697 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2698 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2699 \ifnum#2=\m@ne % Just for the info
2700 \edef\language{tag \bbl@metalang}%
2701 \fi
2702 \bbl@info{\ifnum#2=\m@ne Fetching locale name for tag \bbl@metalang
2703 \else Importing
2704 \ifcase#2font and identification \or basic \fi
2705 data for \language
2706 \fi}%
2707 from babel-#1.ini. Reported}%
2708 \ifnum#2<\@ne
2709 \global\let\bbl@inidata\@empty
2710 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2711 \fi
2712 \def\bbl@section{identification}%
2713 \bbl@exp{%
2714 \\\bbl@inistore tag.ini=#1\\ \@
2715 \\\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\ \@}%
2716 \bbl@loop@ini\bbl@readstream
2717 % == Process stored data ==
2718 \ifnum#2=\m@ne
2719 \def\bbl@tempa##1 ##2\@{##1}% Get first name
2720 \def\bbl@elt##1##2##3{%
2721 \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2722 {\edef\language{\bbl@tempa##3 \@}%
2723 \bbl@id@assign
2724 \def\bbl@elt####1####2####3{}}%
2725 {}}%
2726 \bbl@inidata
2727 \fi
2728 \bbl@csarg\xdef{lini@\language}{#1}%
2729 \bbl@read@ini@aux
2730 % == 'Export' data ==
2731 \bbl@ini@exports{#2}%
2732 \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2733 \global\let\bbl@inidata\@empty
2734 \bbl@exp{\\ \bbl@add@list\\ \bbl@ini@loaded{\language}}%
2735 \bbl@tglobal\bbl@ini@loaded
2736 \fi
2737 \closein\bbl@readstream}
2738 \def\bbl@read@ini@aux{%
2739 \let\bbl@savestrings\@empty
2740 \let\bbl@savetoday\@empty
2741 \let\bbl@savestate\@empty
2742 \def\bbl@elt##1##2##3{%
2743 \def\bbl@section{##1}%
2744 \in@{=date.}{=##1}% Find a better place
2745 \ifin@
2746 \bbl@ifunset{bbl@inikv{##1}}%
2747 {\bbl@ini@calendar{##1}}%
2748 {}}%

```



```

2749 \fi
2750 \bbl@ifunset{bbl@inikv@##1}{}%
2751 {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2752 \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.

```

2753 \def\bbl@extend@ini@aux#1{%
2754 \bbl@startcommands*{#1}{captions}%
2755 % Activate captions/... and modify exports
2756 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2757 \setlocalecaption{#1}{##1}{##2}}}%
2758 \def\bbl@inikv@captions##1##2{%
2759 \bbl@ini@captions@aux{##1}{##2}}}%
2760 \def\bbl@stringdef##1##2{\gdef##1{##2}}}%
2761 \def\bbl@exportkey##1##2##3{%
2762 \bbl@ifunset{bbl@kv@##2}{}%
2763 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2764 \bbl@exp{\global\let\<bbl@##1@language\>\<bbl@kv@##2>}}}%
2765 \fi}}}%
2766 % As with \bbl@read@ini, but with some changes
2767 \bbl@read@ini@aux
2768 \bbl@ini@exports\tw@
2769 % Update inidata@lang by pretending the ini is read.
2770 \def\bbl@elt##1##2##3{%
2771 \def\bbl@section{##1}%
2772 \bbl@iniline##2=##3\bbl@iniline}%
2773 \csname bbl@inidata@#1\endcsname
2774 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2775 \StartBabelCommands*{#1}{date}% And from the import stuff
2776 \def\bbl@stringdef##1##2{\gdef##1{##2}}}%
2777 \bbl@savetoday
2778 \bbl@savestate
2779 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2780 \def\bbl@ini@calendar#1{%
2781 \lowercase{\def\bbl@tempa{=#1=}}}%
2782 \bbl@replace\bbl@tempa{=date.gregorian}{}}}%
2783 \bbl@replace\bbl@tempa{=date.}{}}}%
2784 \in@{.licr=}{#1=}%
2785 \ifin@
2786 \ifcase\bbl@engine
2787 \bbl@replace\bbl@tempa{.licr=}{}}}%
2788 \else
2789 \let\bbl@tempa\relax
2790 \fi
2791 \fi
2792 \ifx\bbl@tempa\relax\else
2793 \bbl@replace\bbl@tempa{=}{}}}%
2794 \ifx\bbl@tempa\@empty\else
2795 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2796 \fi
2797 \bbl@exp{%
2798 \def\<bbl@inikv@#1>####1####2{%
2799 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2800 \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).

```

2801 \def\bbl@renewinikey#1/#2\@#3{%
2802 \global\let\bbl@extend@ini\bbl@extend@ini@aux

```

```

2803 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2804 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2805 \bbl@trim\toks@{#3}% value
2806 \bbl@exp{%
2807 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2808 \\g@addto@macro\\bbl@inidata{%
2809 \\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.

```

2810 \def\bbl@exportkey#1#2#3{%
2811 \bbl@ifunset{\bbl@kv@#2}%
2812 {\bbl@csarg\gdef{#1@\language\language}\{#3}}%
2813 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
2814 \bbl@csarg\gdef{#1@\language\language}\{#3}}%
2815 \else
2816 \bbl@exp{\global\let<\bbl@#1@\language\language>\<\bbl@kv@#2>}%
2817 \fi}}

```

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

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

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

```

2818 \def\bbl@iniwarning#1{%
2819 \bbl@ifunset{\bbl@kv@identification.warning#1}{}}%
2820 {\bbl@warning{%
2821 From babel-\bbl@cs{lini@\language\language}.ini:\\%
2822 \bbl@cs{@kv@identification.warning#1}\\%
2823 Reported}}}
2824 %
2825 \let\bbl@release@transforms\@empty
2826 \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).

```

2827 \def\bbl@ini@exports#1{%
2828 % Identification always exported
2829 \bbl@iniwarning{}%
2830 \ifcase\bbl@engine
2831 \bbl@iniwarning{.pdf\latex}%
2832 \or
2833 \bbl@iniwarning{.lua\latex}%
2834 \or
2835 \bbl@iniwarning{.xel\latex}%
2836 \fi%
2837 \bbl@exportkey{lllevel}{identification.load.level}{}}%
2838 \bbl@exportkey{elname}{identification.name.english}{}}%
2839 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2840 {\csname\bbl@elname@\language\language\endcsname}}%
2841 \bbl@exportkey{tbcp}{identification.tag.bcp47}{}}%
2842 \bbl@exportkey{casing}{identification.tag.bcp47}{}}%
2843 \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}}%
2844 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2845 \bbl@exportkey{esname}{identification.script.name}{}}%

```

```

2846 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2847   {\csname bbl@esname@language\endcsname}}%
2848 \bbl@exportkey{sbc}{identification.script.tag.bcp47}}}%
2849 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2850 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}}}%
2851 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}}}%
2852 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}}}%
2853 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}}}%
2854 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}}}%
2855 % Also maps bcp47 -> language
2856 \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
2857 \ifcase\bbl@engine\or
2858   \directlua{%
2859     Babel.locale_props[\the\bbl@cs{id@language}].script
2860     = '\bbl@cl{sbc}}'}%
2861 \fi
2862 % Conditional
2863 \ifnum#1>z@ % -1 or 0 = only info, 1 = basic, 2 = (re)new
2864 \bbl@exportkey{calpr}{date.calendar.preferred}}}%
2865 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2866 \bbl@exportkey{hyphr}{typography.hyphenrules}}}%
2867 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2868 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2869 \bbl@exportkey{prehc}{typography.prehyphenchar}}}%
2870 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}}}%
2871 \bbl@exportkey{hyots}{typography.hyphenate.other.script}}}%
2872 \bbl@exportkey{intsp}{typography.intraspace}}}%
2873 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2874 \bbl@exportkey{chrng}{characters.ranges}}}%
2875 \bbl@exportkey{quote}{characters.delimiters.quotes}}}%
2876 \bbl@exportkey{dgnat}{numbers.digits.native}}}%
2877 \ifnum#1=\tw@ % only (re)new
2878 \bbl@exportkey{rqtex}{identification.require.babel}}}%
2879 \bbl@tglobal\bbl@savetoday
2880 \bbl@tglobal\bbl@savestate
2881 \bbl@savestrings
2882 \fi
2883 \fi}

```

4.20. Processing keys in ini

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

```

2884 \def\bbl@inikv#1#2{%      key=value
2885   \toks@{#2}%             This hides #'s from ini values
2886   \bbl@csarg\xdef{kv@bbl@section.#1}{\the\toks@}}

```

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

```

2887 \let\bbl@inikv@identification\bbl@inikv
2888 \let\bbl@inikv@date\bbl@inikv
2889 \let\bbl@inikv@typography\bbl@inikv
2890 \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.

```

2891 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@language}\empty x-\fi}
2892 \def\bbl@inikv@characters#1#2{%
2893   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2894   {\bbl@exp{%
2895     \\g@addto@macro\\bbl@release@casing{%
2896       \\bbl@casemapping}{\language}{\unexpanded{#2}}}%
2897     {\in@{casing.}{#1}% e.g., casing.Uv = uV
2898     \ifin@

```

```

2899 \lowercase{\def\bbl@tempb{#1}}%
2900 \bbl@replace\bbl@tempb{casing.}{}%
2901 \bbl@exp{\g@addto@macro{\bbl@release@casing}%
2902 \bbl@casemapping
2903 {\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2904 \else
2905 \bbl@inikv{#1}{#2}%
2906 \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’.

```

2907 \def\bbl@inikv@counters#1#2{%
2908 \bbl@ifsamestring{#1}{digits}%
2909 {\bbl@error{digits-is-reserved}}{}}}%
2910 {}%
2911 \def\bbl@tempc{#1}%
2912 \bbl@trim\def{\bbl@tempb*}{#2}%
2913 \in@{.1$}{#1$}%
2914 \ifin@
2915 \bbl@replace\bbl@tempc{.1}{}%
2916 \bbl@csarg\protected@xdef{cnt@#1\bbl@tempc @\language}{%
2917 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2918 \fi
2919 \in@{.F.}{#1}%
2920 \ifin@else\in@{.S.}{#1}\fi
2921 \ifin@
2922 \bbl@csarg\protected@xdef{cnt@#1@\language}{\bbl@tempb*}%
2923 \else
2924 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2925 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2926 \bbl@csarg{\global\expandafter\let}{cnt@#1@\language}\bbl@tempa
2927 \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.

```

2928 \ifcase\bbl@engine
2929 \bbl@csarg\def{inikv@captions.licr}#1#2{%
2930 \bbl@ini@captions@aux{#1}{#2}}
2931 \else
2932 \def\bbl@inikv@captions#1#2{%
2933 \bbl@ini@captions@aux{#1}{#2}}
2934 \fi

```

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

```

2935 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2936 \bbl@replace\bbl@tempa{.template}{}%
2937 \def\bbl@toreplace{#1}}}%
2938 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}}%
2939 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
2940 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2941 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}}%
2942 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}}%
2943 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2944 \ifin@
2945 \@nameuse{\bbl@patch\bbl@tempa}%
2946 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2947 \fi
2948 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2949 \ifin@
2950 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2951 \bbl@exp{\gdef<fnum@\bbl@tempa>{%
2952 \bbl@ifunset{\bbl@tempa fmt@\language}%

```

```

2953      {\fnum@bbl@tempa}}%
2954      {\@nameuse{bbl@bbl@tempa fmt@\\language}}}%
2955 \fi}
2956 %
2957 \def\bbl@ini@captions@aux#1#2{%
2958   \bbl@trim@def\bbl@tempa{#1}%
2959   \bbl@xin@{.template}{\bbl@tempa}%
2960   \ifin@
2961     \bbl@ini@captions@template{#2}\language
2962   \else
2963     \bbl@ifblank{#2}%
2964     {\bbl@exp{%
2965       \toks@{\@nameuse{bbl@nocaption}{\bbl@tempa name}{\language\bbl@tempa name}}}%
2966       {\bbl@trim\toks@{#2}}}%
2967     \bbl@exp{%
2968       \@nameuse{bbl@add}\bbl@savestrings{%
2969         \SetString<\bbl@tempa name>{\the\toks@}}%
2970       \toks@expandafter{\bbl@captionslist}%
2971       \bbl@exp{\@nameuse{<\bbl@tempa name>}{\the\toks@}}%
2972       \ifin@else
2973         \bbl@exp{%
2974           \@nameuse{bbl@add}<\bbl@extracaps@language>{\<\bbl@tempa name>}%
2975           \@nameuse{bbl@tglobal}<\bbl@extracaps@language>}%
2976       \fi
2977 \fi}

```

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

```

2978 \def\bbl@list@the{%
2979   part,chapter,section,subsection,subsubsection,paragraph,%
2980   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2981   table,page,footnote,mpfootnote,mpfn}
2982 %
2983 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2984   \bbl@ifunset{bbl@map@#1@language}%
2985   {\@nameuse{#1}}%
2986   {\@nameuse{bbl@map@#1@language}}}
2987 %
2988 \def\bbl@map@lbl#1{% #1:a sign, eg, .
2989   \ifincsname#1\else
2990     \bbl@ifunset{bbl@map@#1@language}%
2991     {#1}%
2992     {\@nameuse{bbl@map@#1@language}}%
2993   \fi}
2994 %
2995 \def\bbl@inikv@labels#1#2{%
2996   \in@{.map}{#1}%
2997   \ifin@
2998     \in@{,dot.map,}{, #1,}%
2999   \ifin@
3000     \global\@namedef{bbl@map@.@@@language}{#2}%
3001   \fi
3002   \ifx\bbl@KVP@labels\@nnil\else
3003     \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3004     \ifin@
3005       \def\bbl@tempc{#1}%
3006       \bbl@replace\bbl@tempc{.map}{}%
3007       \in@{, #2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3008       \bbl@exp{%
3009         \gdef<\bbl@map@bbl@tempc @language>%
3010         {\ifin@<#2>\else\\loalecounter{#2}\fi}}%
3011       \bbl@foreach\bbl@list@the{%
3012         \bbl@ifunset{the##1}{%
3013           {\bbl@ncarg\let\bbl@tempd{the##1}%

```

```

3014 \bbl@exp{%
3015 \\\bbl@sreplace\<the##1>%
3016 {\<\bbl@tempc>{##1}}%
3017 {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3018 \\\bbl@sreplace\<the##1>%
3019 {\<\@empty @\bbl@tempc>\<c@##1>%
3020 {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3021 \\\bbl@sreplace\<the##1>%
3022 {\\\csname @\bbl@tempc\\endcsname\<c@##1>%
3023 {\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3024 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3025 \bbl@exp{\gdef\<the##1>{\[the##1]}}%
3026 \fi}%
3027 \fi
3028 \fi
3029 %
3030 \else
3031 % The following code is still under study. You can test it and make
3032 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3033 % language dependent.
3034 \in@{enumerate.}{#1}%
3035 \ifin@
3036 \def\bbl@tempa{#1}%
3037 \bbl@replace\bbl@tempa{enumerate.}{}%
3038 \def\bbl@toreplace{#2}%
3039 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3040 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3041 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3042 \toks@{\expandafter\bbl@toreplace}%
3043 \bbl@exp{%
3044 \\\bbl@add\<extras\language>{%
3045 \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3046 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3047 \\\bbl@toggle\<extras\language>}%
3048 \fi
3049 \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.

```

3050 \def\bbl@chapttype{chapter}
3051 \ifx\@makechapterhead\undefined
3052 \let\bbl@patchchapter\relax
3053 \else\ifx\thechapter\undefined
3054 \let\bbl@patchchapter\relax
3055 \else\ifx\ps@headings\undefined
3056 \let\bbl@patchchapter\relax
3057 \else
3058 \def\bbl@patchchapter{%
3059 \global\let\bbl@patchchapter\relax
3060 \gdef\bbl@chfmt{%
3061 \bbl@ifunset{\bbl@\bbl@chapttype fmt@\language}%
3062 {\@chapapp\space\thechapter}%
3063 {\@nameuse{\bbl@\bbl@chapttype fmt@\language}}}%
3064 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3065 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3066 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3067 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3068 \bbl@toggle\appendix
3069 \bbl@toggle\ps@headings
3070 \bbl@toggle\chaptermark
3071 \bbl@toggle\@makechapterhead}

```

```

3072 \let\bbl@patchappendix\bbl@patchchapter
3073 \fi\fi\fi
3074 \ifx\@part\@undefined
3075 \let\bbl@patchpart\relax
3076 \else
3077 \def\bbl@patchpart{%
3078 \global\let\bbl@patchpart\relax
3079 \gdef\bbl@partformat{%
3080 \bbl@ifunset\bbl@partfmt@\language\name}%
3081 {\partname\nobreakspace\thepart}%
3082 {\@nameuse\bbl@partfmt@\language\name}}}%
3083 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3084 \bbl@tglobal\@part}
3085 \fi

```

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

```

3086 \let\bbl@calendar\@empty
3087 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3088 \def\bbl@localedate#1#2#3#4{%
3089 \begingroup
3090 \edef\bbl@they{#2}%
3091 \edef\bbl@them{#3}%
3092 \edef\bbl@thed{#4}%
3093 \edef\bbl@tempe{%
3094 \bbl@ifunset\bbl@calpr@\language\name}{\bbl@cl{calpr}},%
3095 #1}%
3096 \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3097 \bbl@replace\bbl@tempe{ }{}%
3098 \bbl@replace\bbl@tempe{convert}{convert=}%
3099 \let\bbl@ld@calendar\@empty
3100 \let\bbl@ld@variant\@empty
3101 \let\bbl@ld@convert\relax
3102 \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld@##1}{##2}}%
3103 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3104 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3105 \ifx\bbl@ld@calendar\@empty\else
3106 \ifx\bbl@ld@convert\relax\else
3107 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3108 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3109 \fi
3110 \fi
3111 \@nameuse\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3112 \edef\bbl@calendar{% Used in \month..., too
3113 \bbl@ld@calendar
3114 \ifx\bbl@ld@variant\@empty\else
3115 .\bbl@ld@variant
3116 \fi}%
3117 \bbl@cased
3118 {\@nameuse\bbl@date@\language\name @\bbl@calendar}%
3119 \bbl@they\bbl@them\bbl@thed}%
3120 \endgroup}
3121 %
3122 \def\bbl@printdate#1{%
3123 \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}}
3124 \def\bbl@printdate@i#1[#2]#3#4#5{%
3125 \bbl@usedategroupttrue
3126 \@nameuse\bbl@ensure@#1}{\localedate[#2][#3][#4][#5]}
3127 %
3128 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3129 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3130 \bbl@trim\def\bbl@tempa{#1.#2}%
3131 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate

```

```

3132 {\bbl@trim@def\bbl@tempa{#3}%
3133 \bbl@trim\toks@{#5}%
3134 \@temptokena\expandafter{\bbl@savestate}%
3135 \bbl@exp{% Reverse order - in ini last wins
3136 \def\\bbl@savestate{%
3137 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3138 \the\@temptokena}}}%
3139 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3140 {\lowercase{\def\bbl@tempb{#6}}}%
3141 \bbl@trim@def\bbl@toreplace{#5}%
3142 \bbl@TG@@date
3143 \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3144 \ifx\bbl@savestate@empty
3145 \bbl@exp{%
3146 \\AfterBabelCommands{%
3147 \gdef\<\language name date>{\\protect\<\language name date >}%
3148 \gdef\<\language name date >{\\bbl@printdate{\language name}}}%
3149 \def\\bbl@savestate{%
3150 \\SetString\\today{%
3151 \<\language name date>[convert]%
3152 {\the\year}{\the\month}{\the\day}}}%
3153 \fi}%
3154 {}}}}

```

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).

```

3155 \let\bbl@calendar@empty
3156 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3157 \nameuse{bbl@ca@#2}#1@@}
3158 \newcommand\babelDateSpace{\nobreakspace}
3159 \newcommand\babelDateDot{.\@}
3160 \newcommand\babelDated[1]{\number#1}
3161 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3162 \newcommand\babelDateM[1]{\number#1}
3163 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3164 \newcommand\babelDateMMM[1]{%
3165 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3166 \newcommand\babelDatey[1]{\number#1}%
3167 \newcommand\babelDateyy[1]{%
3168 \ifnum#1<10 0\number#1 %
3169 \else\ifnum#1<100 \number#1 %
3170 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3171 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3172 \else
3173 \bbl@error{limit-two-digits}{\number#1}%
3174 \fi\fi\fi\fi}
3175 \newcommand\babelDateyyyy[1]{\number#1}
3176 \newcommand\babelDateU[1]{\number#1}%
3177 \def\bbl@replace@finish@iii#1{%
3178 \bbl@exp{\def\\#1###1###2###3{\the\toks@}}
3179 \def\bbl@TG@@date{%
3180 \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3181 \bbl@replace\bbl@toreplace{[. ]}{\babelDateDot}}%
3182 \bbl@replace\bbl@toreplace{[d]}{\babelDated{###3}}%
3183 \bbl@replace\bbl@toreplace{[dd]}{\babelDatedd{###3}}%
3184 \bbl@replace\bbl@toreplace{[M]}{\babelDateM{###2}}%
3185 \bbl@replace\bbl@toreplace{[MM]}{\babelDateMM{###2}}%
3186 \bbl@replace\bbl@toreplace{[MMM]}{\babelDateMMM{###2}}%
3187 \bbl@replace\bbl@toreplace{[y]}{\babelDatey{###1}}%
3188 \bbl@replace\bbl@toreplace{[yy]}{\babelDateyy{###1}}%

```



```

3189 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}}%
3190 \bbl@replace\bbl@toreplace{[U]}\BabelDateU{####1}}%
3191 \bbl@replace\bbl@toreplace{[y]}\bbl@datecctr{####1}}%
3192 \bbl@replace\bbl@toreplace{[U]}\bbl@datecctr{####1}}%
3193 \bbl@replace\bbl@toreplace{[m]}\bbl@datecctr{####2}}%
3194 \bbl@replace\bbl@toreplace{[d]}\bbl@datecctr{####3}}%
3195 \bbl@replace@finish@iii\bbl@toreplace}
3196 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3197 \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.

```

3198 \AddToHook{begindocument/before}{%
3199 \let\bbl@normalsf\normalsfcodes
3200 \let\normalsfcodes\relax}
3201 \AtBeginDocument{%
3202 \ifx\bbl@normalsf\@empty
3203 \ifnum\sfcodes`\.=\@m
3204 \let\normalsfcodes\frenchspacing
3205 \else
3206 \let\normalsfcodes\nonfrenchspacing
3207 \fi
3208 \else
3209 \let\normalsfcodes\bbl@normalsf
3210 \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.

```

3211 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3212 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3213 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3214 #1[#2]{#3}{#4}{#5}}
3215 \begingroup
3216 \catcode`\%=12
3217 \catcode`\&=14
3218 \gdef\bbl@transforms#1#2#3{%&
3219 \directlua{
3220 local str = [=[#2]=]
3221 str = str:gsub('%.%d+%.%d+$', '')
3222 token.set_macro('babeltempa', str)
3223 }&
3224 \def\babeltempc{}&
3225 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&
3226 \ifin@else
3227 \bbl@xin@{:,\babeltempa,}{,\bbl@KVP@transforms,}&
3228 \fi
3229 \ifin@
3230 \bbl@foreach\bbl@KVP@transforms{%&
3231 \bbl@xin@{:,\babeltempa,}{,##1,}&
3232 \ifin@ & font:font:transform syntax
3233 \directlua{
3234 local t = {}
3235 for m in string.gmatch('##1'..' ':'', '(.)') do
3236 table.insert(t, m)
3237 end
3238 table.remove(t)
3239 token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))

```

```

3240         }&%
3241     \fi}&%
3242     \in@{.0$}{#2$}&%
3243     \ifin@
3244         \directlua{&% (\attribute) syntax
3245             local str = string.match([[ \bbl@KVP@transforms]],
3246                 '%([^(%[-])%)[^%)]-\babeltempa')
3247             if str == nil then
3248                 token.set_macro('babeltempb', '')
3249             else
3250                 token.set_macro('babeltempb', ',attribute=' .. str)
3251             end
3252         }&%
3253     \toks@{#3}&%
3254     \bbl@exp{&%
3255         \\g@addto@macro\\bbl@release@transforms{&%
3256             \relax &% Closes previous \bbl@transforms@aux
3257             \\bbl@transforms@aux
3258             \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3259             {\language\the\toks@}}&%
3260     \else
3261         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3262     \fi
3263 \fi}
3264 \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.

```

3265 \def\bbl@provide@lsys#1{%
3266     \bbl@ifunset{bbl@lname@#1}%
3267         {\bbl@load@info{#1}}%
3268     }%
3269     \bbl@csarg\let{lsys@#1}\empty
3270     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3271     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3272     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3273     \bbl@ifunset{bbl@lname@#1}{}%
3274         {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3275     \ifcase\bbl@engine\or\or
3276         \bbl@ifunset{bbl@prehc@#1}{}%
3277             {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3278                 }%
3279             {\ifx\bbl@xenoxyph\undefined
3280                 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3281                 \ifx\AtBeginDocument\@notprerr
3282                     \expandafter\@secondoftwo % to execute right now
3283                 \fi
3284                 \AtBeginDocument{%
3285                     \bbl@patchfont{\bbl@xenoxyph}%
3286                     {\expandafter\select@language\expandafter{\language\the\toks@}}}%
3287             \fi}}%
3288     \fi
3289     \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.

[illegible]

```

3321 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3322 \ifx\\#1% % \\ before, in case #1 is multiletter
3323 \bbl@exp{%
3324 \def\\bbl@tempa####1{%
3325 \<ifcase>####1\space\the\toks@\<else>\\@ctrerrr\<fi>}}%
3326 \else
3327 \toks@\expandafter{\the\toks@\or #1}%
3328 \expandafter\bbl@buildifcase
3329 \fi}

```

```

3330 \newcommand\localenumberal[2]{\bbl@cs{cntnr#1@{language name}}{#2}}
3331 \def\bbl@localecntnr#1#2{\localenumberal{#2}{#1}}
3332 \newcommand\localecounter[2]{%
3333   \expandafter\bbl@localecntnr
3334   \expandafter{\number\csname c@#2\endcsname}{#1}}
3335 \def\bbl@alphnumerical#1#2{%
3336   \expandafter\bbl@alphnumerical@i\number#2 76543210@@{#1}}
3337 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8@@@#9{%
3338   \ifcase\@car#8\@nil\or    % Currently <10000, but prepared for bigger
3339     \bbl@alphnumerical@ii{#9}000000#1\or
3340     \bbl@alphnumerical@ii{#9}00000#1#2\or
3341     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3342     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3343     \bbl@alphnum@invalid{>9999}%

```

```

3344 \fi}
3345 \def\bbl@alphanumeric@iii#1#2#3#4#5#6#7#8{%
3346 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@{language}}%
3347 {\bbl@cs{cntr@#1.4@{language}}#5%
3348 \bbl@cs{cntr@#1.3@{language}}#6%
3349 \bbl@cs{cntr@#1.2@{language}}#7%
3350 \bbl@cs{cntr@#1.1@{language}}#8%
3351 \ifnum#6#7#8>\z@
3352 \bbl@ifunset{bbl@cntr@#1.S.321@{language}}{%
3353 {\bbl@cs{cntr@#1.S.321@{language}}}%
3354 \fi}%
3355 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@{language}}}}
3356 \def\bbl@alphnum@invalid#1{%
3357 \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```

3358 \newcommand\BabelUppercaseMapping[3]{%
3359 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3360 \newcommand\BabelTitlecaseMapping[3]{%
3361 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3362 \newcommand\BabelLowercaseMapping[3]{%
3363 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

  The parser for casing and casing.<variant>.
3364 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3365 \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3366 \else
3367 \def\bbl@uftocode#1{\expandafter`\string#1}
3368 \fi
3369 \def\bbl@casemapping#1#2#3{% 1:variant
3370 \def\bbl@tempa##1 ##2{% Loop
3371 \bbl@casemapping@i{##1}%
3372 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3373 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3374 \def\bbl@tempe{0}% Mode (upper/lower...)
3375 \def\bbl@tempc{#3}% Casing list
3376 \expandafter\bbl@tempa\bbl@tempc\@empty}
3377 \def\bbl@casemapping@i#1{%
3378 \def\bbl@tempb{#1}%
3379 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3380 \@nameuse{regex_replace_all:nnN}%
3381 {[\\x{c0}-\\x{ff}][\\x{80}-\\x{bf}]*}{\\0}}\bbl@tempb
3382 \else
3383 \@nameuse{regex_replace_all:nnN}{.}{\\0}}\bbl@tempb
3384 \fi
3385 \expandafter\bbl@casemapping@ii\bbl@tempb\@
3386 \def\bbl@casemapping@ii#1#2#3\@@{%
3387 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3388 \ifin@
3389 \edef\bbl@tempe{%
3390 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3391 \else
3392 \ifcase\bbl@tempe\relax
3393 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3394 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3395 \or
3396 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3397 \or
3398 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3399 \or
3400 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3401 \fi
3402 \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.

```
3403 \def\bbl@localeinfo#1#2{%
3404   \bbl@ifunset{\bbl@info@#2}{#1}%
3405   {\bbl@ifunset{\bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3406    {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3407 \newcommand\localeinfo[1]{%
3408   \ifx*#1\@empty
3409     \bbl@afterelse\bbl@localeinfo{%
3410   \else
3411     \bbl@localeinfo
3412     {\bbl@error{no-ini-info}{}}{}}}%
3413   {#1}%
3414   \fi}
3415 % \@namedef{\bbl@info@name.locale}{lcname}
3416 \@namedef{\bbl@info@tag.ini}{lini}
3417 \@namedef{\bbl@info@name.english}{elname}
3418 \@namedef{\bbl@info@name.opentype}{lname}
3419 \@namedef{\bbl@info@tag.bcp47}{tbc}
3420 \@namedef{\bbl@info@language.tag.bcp47}{lbc}
3421 \@namedef{\bbl@info@tag.opentype}{lotf}
3422 \@namedef{\bbl@info@script.name}{esname}
3423 \@namedef{\bbl@info@script.name.opentype}{sname}
3424 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
3425 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3426 \@namedef{\bbl@info@region.tag.bcp47}{rbcp}
3427 \@namedef{\bbl@info@variant.tag.bcp47}{vbc}
3428 \@namedef{\bbl@info@extension.t.tag.bcp47}{extt}
3429 \@namedef{\bbl@info@extension.u.tag.bcp47}{extu}
3430 \@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.

```
3431 << *More package options >> ≡
3432 \DeclareOption{ensureinfo=off}{}
3433 << /More package options >>
3434 \let\BabelEnsureInfo\relax
```

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

```
3435 \newcommand\getlocaleproperty{%
3436   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3437 \def\bbl@getproperty@s#1#2#3{%
3438   \let#1\relax
3439   \def\bbl@elt##1##2##3{%
3440     \bbl@ifsamestring{##1/##2}{#3}%
3441     {\providecommand#1{##3}%
3442      \def\bbl@elt####1####2####3{}}}%
3443   {}}%
3444   \bbl@cs{inidata@#2}}%
3445 \def\bbl@getproperty@x#1#2#3{%
3446   \bbl@getproperty@s{#1}{#2}{#3}%
3447   \ifx#1\relax
3448     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3449   \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`.

```
3450 \let\bbl@ini@loaded\@empty
3451 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3452 \def\ShowLocaleProperties#1{%
3453   \typeout{}}%
3454   \typeout{*** Properties for language '#1' ***}
```

```

3455 \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3456 \@nameuse{bbl@inidata@#1}%
3457 \typeout{*****}}

```

4.26. BCP 47 related commands

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

```

3458 \newif\ifbbl@bcppallowed
3459 \bbl@bcppallowedfalse
3460 \def\bbl@autoload@options{@import}
3461 \def\bbl@provide@locale{%
3462   \ifx\babelprovide\@undefined
3463     \bbl@error{base-on-the-fly}{}}}%
3464 \fi
3465 \let\bbl@auxname\language
3466 \ifbbl@bcptoname
3467   \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
3468   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
3469   \let\locale\language}%
3470 \fi
3471 \ifbbl@bcppallowed
3472   \expandafter\ifx\csname date\language\endcsname\relax
3473     \expandafter
3474     \bbl@bcplookup\language-\@empty-\@empty-\@empty@@
3475     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3476       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3477       \let\locale\language
3478       \expandafter\ifx\csname date\language\endcsname\relax
3479         \let\bbl@initoload\bbl@bcp
3480         \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\language}}%
3481         \let\bbl@initoload\relax
3482       \fi
3483       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\locale}%
3484     \fi
3485   \fi
3486 \fi
3487 \expandafter\ifx\csname date\language\endcsname\relax
3488   \IfFileExists{babel-\language.tex}%
3489   {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3490   {}%
3491 \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.`<s>` for singletons may change.

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

```

3492 \providecommand\BCPdata{}
3493 \ifx\renewcommand\@undefined\else
3494   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3495   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3496     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3497     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3498     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3499   \def\bbl@bcpdata@ii#1#2{%
3500     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3501     {\bbl@error{unknown-ini-field}{#1}{}}}%

```

```

3502      {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3503      {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3504 \fi
3505 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3506 \@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.

```

3507 \newcommand\babeladjust[1]{%
3508   \bbl@forkv{#1}{%
3509     \bbl@ifunset{bbl@ADJ@##1@##2}%
3510     {\bbl@cs{ADJ@##1}{##2}}%
3511     {\bbl@cs{ADJ@##1@##2}}}
3512 %
3513 \def\bbl@adjust@lua#1#2{%
3514   \ifvmode
3515     \ifnum\currentgrouplevel=\z@
3516       \directlua{ Babel.#2 }%
3517       \expandafter\expandafter\expandafter@gobble
3518     \fi
3519   \fi
3520   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3521 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3522   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3523 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3524   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3525 \@namedef{bbl@ADJ@bidi.text@on}{%
3526   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3527 \@namedef{bbl@ADJ@bidi.text@off}{%
3528   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3529 \@namedef{bbl@ADJ@bidi.math@on}{%
3530   \let\bbl@noamsmath\empty}
3531 \@namedef{bbl@ADJ@bidi.math@off}{%
3532   \let\bbl@noamsmath\relax}
3533 %
3534 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3535   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3536 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3537   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3538 %
3539 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3540   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3541 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3542   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3543 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3544   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3545 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3546   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3547 \@namedef{bbl@ADJ@justify.arabic@on}{%
3548   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3549 \@namedef{bbl@ADJ@justify.arabic@off}{%
3550   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3551 %
3552 \def\bbl@adjust@layout#1{%
3553   \ifvmode
3554     #1%
3555     \expandafter\expandafter\expandafter@gobble
3556   \fi
3557   {\bbl@error{layout-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3558 \@namedef{bbl@ADJ@layout.tabular@on}{%
3559   \ifnum\bbl@tabular@mode=\tw@

```

```

3560 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3561 \else
3562 \chardef\bbl@tabular@mode\@ne
3563 \fi}
3564 \@namedef{bbl@ADJ@layout.tabular@off}{%
3565 \ifnum\bbl@tabular@mode=\tw@
3566 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3567 \else
3568 \chardef\bbl@tabular@mode\z@
3569 \fi}
3570 \@namedef{bbl@ADJ@layout.lists@on}{%
3571 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3572 \@namedef{bbl@ADJ@layout.lists@off}{%
3573 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3574 %
3575 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3576 \bbl@bcpallowedtrue}
3577 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3578 \bbl@bcpallowedfalse}
3579 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3580 \def\bbl@bcp@prefix{#1}}
3581 \def\bbl@bcp@prefix{bcp47-}
3582 \@namedef{bbl@ADJ@autoload.options#1}{%
3583 \def\bbl@autoload@options{#1}}
3584 \def\bbl@autoload@bcptoptions{import}
3585 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3586 \def\bbl@autoload@bcptoptions{#1}}
3587 \newif\ifbbl@bcptname
3588 %
3589 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3590 \bbl@bcptnametrue}
3591 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3592 \bbl@bcptnamefalse}
3593 %
3594 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3595 \directlua{ Babel.ignore_pre_char = function(node)
3596 return (node.lang == \the\csname l@nohyphenation\endcsname)
3597 end }}
3598 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3599 \directlua{ Babel.ignore_pre_char = function(node)
3600 return false
3601 end }}
3602 %
3603 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3604 \def\bbl@ignoreinterchar{%
3605 \ifnum\language=\l@nohyphenation
3606 \expandafter\@gobble
3607 \else
3608 \expandafter\@firstofone
3609 \fi}}
3610 \@namedef{bbl@ADJ@interchar.disable@off}{%
3611 \let\bbl@ignoreinterchar\@firstofone}
3612 %
3613 \@namedef{bbl@ADJ@select.write@shift}{%
3614 \let\bbl@restorelastskip\relax
3615 \def\bbl@savelastskip{%
3616 \let\bbl@restorelastskip\relax
3617 \ifvmode
3618 \ifdim\lastskip=\z@
3619 \let\bbl@restorelastskip\nobreak
3620 \else
3621 \bbl@exp{%
3622 \def\\bbl@restorelastskip%

```



```

3623         \skip@=\the\lastskip
3624         \\nobreak \vskip-\skip@ \vskip\skip@}}%
3625     \fi
3626 \fi}}
3627 \@namedef{bbl@ADJ@select.write@keep}{%
3628     \let\bbl@restorelastskip\relax
3629     \let\bbl@savelastskip\relax}
3630 \@namedef{bbl@ADJ@select.write@omit}{%
3631     \AddBabelHook{babel-select}{beforestart}{%
3632         \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3633     \let\bbl@restorelastskip\relax
3634     \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3635 \@namedef{bbl@ADJ@select.encoding@off}{%
3636     \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.

```

3637 << *More package options >> ≡
3638 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3639 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3640 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3641 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3642 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3643 << /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.

```

3644 \bbl@trace{Cross referencing macros}
3645 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3646     \def\@newl@bel#1#2#3{%
3647         {\@safe@activetrue
3648         \bbl@ifunset{#1@#2}%
3649             \relax
3650             {\gdef\@multiplelabels{%
3651                 \latex@warning@no@line{There were multiply-defined labels}}}%
3652                 \latex@warning@no@line{Label `#2' multiply defined}}}%
3653     \global\@namedef{#1@#2}{#3}}

```

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

```

3654 \CheckCommand*\@testdef[3]{%
3655     \def\reserved@a{#3}%
3656     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3657     \else
3658         \@tempwattrue
3659     \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.

```

3660 \def\@testdef#1#2#3{%
3661   \@safe@activetrue
3662   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3663   \def\bbl@tempb{#3}%
3664   \@safe@activetrue
3665   \ifx\bbl@tempa\relax
3666   \else
3667     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3668   \fi
3669   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3670   \ifx\bbl@tempa\bbl@tempb
3671   \else
3672     \@tempwattrue
3673   \fi}
3674 \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.

```

3675 \bbl@xin@{R}\bbl@opt@safe
3676 \ifin@
3677   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3678   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3679   {\expandafter\strip@prefix\meaning\ref}%
3680 \ifin@
3681   \bbl@redefine\@kernel@ref#1{%
3682     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3683   \bbl@redefine\@kernel@pageref#1{%
3684     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3685   \bbl@redefine\@kernel@sref#1{%
3686     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3687   \bbl@redefine\@kernel@spageref#1{%
3688     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3689   \else
3690     \bbl@redefineroobust\ref#1{%
3691       \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3692     \bbl@redefineroobust\pageref#1{%
3693       \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3694   \fi
3695 \else
3696   \let\org@ref\ref
3697   \let\org@pageref\pageref
3698 \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.

```

3699 \bbl@xin@{B}\bbl@opt@safe
3700 \ifin@
3701   \bbl@redefine\@citex[#1]#2{%
3702     \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetrue
3703     \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.)

```
3704 \AtBeginDocument{%
3705   \@ifpackageloaded{natbib}{%
3706     \def\@citex[#1][#2]#3{%
3707       \@safe@activestrue\edef\bbl@tempa{#3}\@safe@activesfalse
3708       \org@citex[#1][#2]{\bbl@tempa}}%
3709     }{}}
```

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

```
3710 \AtBeginDocument{%
3711   \@ifpackageloaded{cite}{%
3712     \def\@citex[#1]#2{%
3713       \@safe@activestrue\org@citex[#1][#2]\@safe@activesfalse}%
3714     }{}}
```

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

```
3715 \bbl@redefine\nocite#1{%
3716   \@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.

```
3717 \bbl@redefine\bibcite{%
3718   \bbl@cite@choice
3719   \bibcite}
```

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

```
3720 \def\bbl@bibcite#1#2{%
3721   \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.

```
3722 \def\bbl@cite@choice{%
3723   \global\let\bibcite\bbl@bibcite
3724   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3725     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3726     \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.

```
3727 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3728 \bbl@redefine\@bibitem#1{%
3729   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3730 \else
3731   \let\org@nocite\nocite
3732   \let\org@citex\@citex
```

```

3733 \let\org@bibtex\org@bibtex
3734 \let\org@bibitem\org@bibitem
3735 \fi

```

5.2. Layout

```

3736 \newcommand\BabelPatchSection[1]{%
3737   \@ifundefined{#1}{}{%
3738     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3739     \@namedef{#1}{%
3740       \ifstar{\bbl@presec@#1}%
3741       {\@dblarg{\bbl@presec@x{#1}}}}%
3742 \def\bbl@presec@x#1[#2]#3{%
3743   \bbl@exp{%
3744     \\\select@language@x{\bbl@main@language}%
3745     \\\bbl@cs{sspre@#1}%
3746     \\\bbl@cs{ss@#1}%
3747     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3748     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3749     \\\select@language@x{\language}}%
3750 \def\bbl@presec@#1#2{%
3751   \bbl@exp{%
3752     \\\select@language@x{\bbl@main@language}%
3753     \\\bbl@cs{sspre@#1}%
3754     \\\bbl@cs{ss@#1}*%
3755     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3756     \\\select@language@x{\language}}%
3757 %
3758 \IfBabelLayout{sectioning}%
3759   {\BabelPatchSection{part}%
3760    \BabelPatchSection{chapter}%
3761    \BabelPatchSection{section}%
3762    \BabelPatchSection{subsection}%
3763    \BabelPatchSection{subsubsection}%
3764    \BabelPatchSection{paragraph}%
3765    \BabelPatchSection{subparagraph}%
3766    \def\babel@toc#1{%
3767      \select@language@x{\bbl@main@language}}}%
3768 \IfBabelLayout{captions}%
3769   {\BabelPatchSection{caption}}}%

```

\BabelFootnote Footnotes.

```

3770 \bbl@trace{Footnotes}
3771 \def\bbl@footnote#1#2#3{%
3772   \@ifnextchar[%
3773     {\bbl@footnote@o{#1}{#2}{#3}}%
3774     {\bbl@footnote@x{#1}{#2}{#3}}%
3775 \long\def\bbl@footnote@x#1#2#3#4{%
3776   \bgroup
3777   \select@language@x{\bbl@main@language}%
3778   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3779   \egroup}
3780 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3781   \bgroup
3782   \select@language@x{\bbl@main@language}%
3783   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3784   \egroup}
3785 \def\bbl@footnotetext#1#2#3{%
3786   \@ifnextchar[%
3787     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3788     {\bbl@footnotetext@x{#1}{#2}{#3}}%
3789 \long\def\bbl@footnotetext@x#1#2#3#4{%
3790   \bgroup

```

```

3791 \select@language@x{\bbl@main@language}%
3792 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3793 \egroup}
3794 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3795 \bgroup
3796 \select@language@x{\bbl@main@language}%
3797 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3798 \egroup}
3799 \def\BabelFootnote#1#2#3#4{%
3800 \ifx\bbl@fn@footnote\undefined
3801 \let\bbl@fn@footnote\footnote
3802 \fi
3803 \ifx\bbl@fn@footnotetext\undefined
3804 \let\bbl@fn@footnotetext\footnotetext
3805 \fi
3806 \bbl@ifblank{#2}%
3807 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3808 \namedef{\bbl@stripslash#1text}%
3809 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3810 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3811 \namedef{\bbl@stripslash#1text}%
3812 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3813 \IfBabelLayout{footnotes}%
3814 {\let\bbl@OL@footnote\footnote
3815 \BabelFootnote\footnote\language\{}}%
3816 \BabelFootnote\localfootnote\language\{}}%
3817 \BabelFootnote\mainfootnote\{}}%
3818 {}

```

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.

```

3819 \bbl@trace{Marks}
3820 \IfBabelLayout{sectioning}
3821 {\ifx\bbl@opt@headfoot\@nnil
3822 \g@addto@macro\@resetactivechars{%
3823 \set@typeset@protect
3824 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3825 \let\protect\noexpand
3826 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3827 \edef\thepage{%
3828 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3829 \fi}%
3830 \fi}
3831 {\ifbbl@single\else
3832 \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust
3833 \markright#1{%
3834 \bbl@ifblank{#1}%
3835 {\org@markright{}}%
3836 {\toks@{#1}%
3837 \bbl@exp{%
3838 \org@markright{\protect\foreignlanguage{\language}%
3839 {\protect\bbl@restore@actives\the\toks@}}}%

```

\markboth

\@mkboth The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers. The documentclasses report and book define and set the headings for the page.

While doing so they also store a copy of `\markboth` in `\@mkboth`. Therefore we need to check whether `\@mkboth` has already been set. If so we need to do that again with the new definition of `\markboth`. (As of Oct 2019, \TeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3840 \ifx\@mkboth\markboth
3841 \def\bb@tempc{\let\@mkboth\markboth}%
3842 \else
3843 \def\bb@tempc{%
3844 \fi
3845 \bb@ifunset{markboth }\bb@redefine\bb@redefineroobust
3846 \markboth#1#2{%
3847 \protected@edef\bb@tempb##1{%
3848 \protect\foreignlanguage
3849 {\language\name}{\protect\bb@restore@actives##1}}%
3850 \bb@ifblank{#1}%
3851 {\toks@{}}%
3852 {\toks@\expandafter{\bb@tempb{#1}}}%
3853 \bb@ifblank{#2}%
3854 {\@temptokena{}}%
3855 {\@temptokena\expandafter{\bb@tempb{#2}}}%
3856 \bb@exp{\org@markboth{\the\toks@{\the\@temptokena}}}%
3857 \bb@tempc
3858 \fi} % end ifbb@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.

```

3859 \bb@trace{Preventing clashes with other packages}
3860 \ifx\org@ref\undefined\else
3861 \bb@xin@{R}\bb@opt@safe
3862 \ifin@
3863 \AtBeginDocument{%
3864 \@ifpackageloaded{ifthen}{%
3865 \bb@redefine@long\ifthenelse#1#2#3{%
3866 \let\bb@temp@pref\pageref
3867 \let\pageref\org@pageref
3868 \let\bb@temp@ref\ref
3869 \let\ref\org@ref
3870 \@safe@activestrue
3871 \org@ifthenelse{#1}%
3872 {\let\pageref\bb@temp@pref
3873 \let\ref\bb@temp@ref
3874 \@safe@activesfalse
3875 #2}%
3876 {\let\pageref\bb@temp@pref

```

```

3877         \let\ref\bbl@temp@ref
3878         \@safe@activesfalse
3879         #3}%
3880     }%
3881 }{}%
3882 }
3883 \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`.

```

3884 \AtBeginDocument{%
3885   \@ifpackageloaded{varioref}{%
3886     \bbl@redefine\@@vpageref#1[#2]#3{%
3887       \@safe@activestrue
3888       \org@@vpageref{#1}[#2]#3}%
3889     \@safe@activesfalse}%
3890   \bbl@redefine\vrefpagenum#1#2{%
3891     \@safe@activestrue
3892     \org@vrefpagenum{#1}#2}%
3893   \@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.

```

3894   \expandafter\def\csname Ref \endcsname#1{%
3895     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3896   }{}%
3897 }
3898 \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.

```

3899 \AtEndOfPackage{%
3900   \AtBeginDocument{%
3901     \@ifpackageloaded{hhline}%
3902     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3903       \else
3904         \makeatletter
3905         \def\@currname{hhline}\input{hhline.sty}\makeatother
3906       \fi}%
3907     {}}}

```

\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`).

```

3908 \def\substitutefontfamily#1#2#3{%
3909   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3910   \immediate\write15{%
3911     \string\ProvidesFile{#1#2.fd}%
3912     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}}

```

```

3913 \space generated font description file]^^J
3914 \string\DeclareFontFamily{#1}{#2}{ }^^J
3915 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^^J
3916 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^^J
3917 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^^J
3918 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^^J
3919 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^^J
3920 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^^J
3921 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^^J
3922 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^^J
3923 }%
3924 \closeout15
3925 }
3926 \@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`

```

3927 \bbl@trace{Encoding and fonts}
3928 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3929 \newcommand\BabelNonText{TS1,T3,TS3}
3930 \let\org@TeX\TeX
3931 \let\org@LaTeX\LaTeX
3932 \let\ensureascii\@firstofone
3933 \let\asciienencoding\@empty
3934 \AtBeginDocument{%
3935   \def\elt#1{,#1,}%
3936   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3937   \let\elt\relax
3938   \let\bbl@tempb\@empty
3939   \def\bbl@tempc{OT1}%
3940   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3941     \bbl@ifunset{T@#1}{ }\def\bbl@tempb{#1}}}%
3942   \bbl@foreach\bbl@tempa{%
3943     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3944     \ifin@
3945       \def\bbl@tempb{#1}% Store last non-ascii
3946     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3947     \ifin@else
3948       \def\bbl@tempc{#1}% Store last ascii
3949     \fi
3950   \fi}%
3951   \ifx\bbl@tempb\@empty\else
3952     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3953     \ifin@else
3954       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3955     \fi
3956   \let\asciienencoding\bbl@tempc
3957   \renewcommand\ensureascii[1]{%
3958     {\fontencoding{\asciienencoding}\selectfont#1}}%
3959   \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3960   \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3961   \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.

```
3962 \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.

```
3963 \AtBeginDocument{%
3964   \@ifpackageloaded{fontspec}%
3965   {\xdef\latinencoding{%
3966     \ifx\UTFencname\@undefined
3967       EU\ifcase\bbl@engine\or2\or1\fi
3968     \else
3969       \UTFencname
3970     \fi}}%
3971   {\gdef\latinencoding{OT1}%
3972     \ifx\cf@encoding\bbl@t@one
3973       \xdef\latinencoding{\bbl@t@one}%
3974     \else
3975       \def\@elt#1{,#1,}%
3976       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3977       \let\@elt\relax
3978       \bbl@xin@{,T1,}\bbl@tempa
3979       \ifin@
3980         \xdef\latinencoding{\bbl@t@one}%
3981       \fi
3982     \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.

```
3983 \DeclareRobustCommand{\latintext}{%
3984   \fontencoding{\latinencoding}\selectfont
3985   \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.

```
3986 \ifx\@undefined\DeclareTextFontCommand
3987   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3988 \else
3989   \DeclareTextFontCommand{\textlatin}{\latintext}
3990 \fi
```

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

```
3991 \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 T_EX 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 LuaT_EX-jā shows, vertical typesetting is possible, too.

```

3992 \bbl@trace{Loading basic (internal) bidi support}
3993 \ifodd\bbl@engine
3994 \else % Any xe+lua bidi
3995   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3996     \bbl@error{bidi-only-lua}{}}{}%
3997     \let\bbl@beforeforeign\leavevmode
3998     \AtEndOfPackage{%
3999       \EnableBabelHook{babel-bidi}%
4000       \bbl@xebidipar}
4001 \fi\fi
4002 \def\bbl@loadxebidi#1{%
4003   \ifx\RTLfootnotetext\@undefined
4004     \AtEndOfPackage{%
4005       \EnableBabelHook{babel-bidi}%
4006       \ifx\fontspec\@undefined
4007         \usepackage{fontspec}% bidi needs fontspec
4008       \fi
4009       \usepackage#1{bidi}%
4010       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4011       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4012         \ifnum\@nameuse\bbl@wdir\@languagename=\tw@ % 'AL' bidi
4013           \bbl@digitsdotdash % So ignore in 'R' bidi
4014         \fi}}%
4015   \fi}
4016 \ifnum\bbl@bidimode>200 % Any xe bidi=
4017   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4018     \bbl@tentative{bidi=bidi}
4019     \bbl@loadxebidi{}
4020   \or
4021     \bbl@loadxebidi{[rldocument]}
4022   \or
4023     \bbl@loadxebidi{}
4024   \fi
4025 \fi
4026 \fi
4027 \ifnum\bbl@bidimode=\@ne % bidi=default
4028   \let\bbl@beforeforeign\leavevmode
4029   \ifodd\bbl@engine % lua
4030     \newattribute\bbl@attr@dir
4031     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4032     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4033   \fi
4034   \AtEndOfPackage{%
4035     \EnableBabelHook{babel-bidi}% pdf/lua/xe
4036     \ifodd\bbl@engine\else % pdf/xe
4037       \bbl@xebidipar
4038     \fi}
4039 \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.

```

4040 \bbl@trace{Macros to switch the text direction}

```

```

4041 \def\bbl@alscripts{%
4042   ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
4043 \def\bbl@rscripts{%
4044   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4045   Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
4046   Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaean,%
4047   Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
4048   Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
4049   Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
4050   Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
4051   Meroitic,N'Ko,Orkhon,Todhri}
4052 %
4053 \def\bbl@provide@dirs#1{%
4054   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4055   \ifin@
4056     \global\bbl@csarg\chardef{wdir@#1}\@ne
4057     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4058     \ifin@
4059       \global\bbl@csarg\chardef{wdir@#1}\tw@
4060       \fi
4061   \else
4062     \global\bbl@csarg\chardef{wdir@#1}\z@
4063   \fi
4064   \ifodd\bbl@engine
4065     \bbl@csarg\ifcase{wdir@#1}%
4066       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4067     \or
4068       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4069     \or
4070       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4071     \fi
4072   \fi}
4073 %
4074 \def\bbl@switchdir{%
4075   \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4076   \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4077   \bbl@expf{\bbl@setdirs\bbl@cl{wdir}}}%
4078 \def\bbl@setdirs#1{%
4079   \ifcase\bbl@select@type
4080     \bbl@bodydir{#1}%
4081     \bbl@pardir{#1}% <- Must precede \bbl@texdir
4082   \fi
4083   \bbl@texdir{#1}}
4084 \ifnum\bbl@bidimode>\z@
4085   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4086   \DisableBabelHook{babel-bidi}
4087 \fi

```

Now the engine-dependent macros.

```

4088 \ifodd\bbl@engine % luatex=1
4089 \else % pdftex=0, xetex=2
4090   \newcount\bbl@dirlevel
4091   \chardef\bbl@thetexdir\z@
4092   \chardef\bbl@thepardir\z@
4093   \def\bbl@texdir#1{%
4094     \ifcase#1\relax
4095       \chardef\bbl@thetexdir\z@
4096       \@nameuse{setlatin}%
4097       \bbl@texdir@i\beginL\endL
4098     \else
4099       \chardef\bbl@thetexdir\@ne
4100       \@nameuse{setnonlatin}%
4101       \bbl@texdir@i\beginR\endR

```

```

4102 \fi}
4103 \def\bbl@textdir@i#1#2{%
4104 \ifhmode
4105 \ifnum\currentgrouplevel>\z@
4106 \ifnum\currentgrouplevel=\bbl@dirlevel
4107 \bbl@error{multiple-bidi}{\}\}\}%
4108 \bgroup\aftergroup#2\aftergroup\egroup
4109 \else
4110 \ifcase\currentgrouptype\or % 0 bottom
4111 \aftergroup#2% 1 simple {}
4112 \or
4113 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4114 \or
4115 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4116 \or\or\or % vbox vtop align
4117 \or
4118 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4119 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4120 \or
4121 \aftergroup#2% 14 \begingroup
4122 \else
4123 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4124 \fi
4125 \fi
4126 \bbl@dirlevel\currentgrouplevel
4127 \fi
4128 #1%
4129 \fi}
4130 \def\bbl@pdir#1{\chardef\bbl@thepardir#1\relax}
4131 \let\bbl@bodydir\@gobble
4132 \let\bbl@pagedir\@gobble
4133 \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).

```

4134 \def\bbl@xebidipar{%
4135 \let\bbl@xebidipar\relax
4136 \TeXeTstate\@ne
4137 \def\bbl@xeeverypar{%
4138 \ifcase\bbl@thepardir
4139 \ifcase\bbl@thetextdir\else\beginR\fi
4140 \else
4141 {\setbox\z@\lastbox\beginR\box\z@}%
4142 \fi}%
4143 \AddToHook{para/begin}{\bbl@xeeverypar}}
4144 \ifnum\bbl@bidimode>200 % Any xe bidi=
4145 \let\bbl@textdir@i\@gobbletwo
4146 \let\bbl@xebidipar\@empty
4147 \AddBabelHook{bidi}{foreign}{%
4148 \ifcase\bbl@thetextdir
4149 \BabelWrapText{\LR{##1}}%
4150 \else
4151 \BabelWrapText{\RL{##1}}%
4152 \fi}
4153 \def\bbl@pdir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4154 \fi
4155 \fi

```

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

```

4156 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}}
4157 \AtBeginDocument{%
4158 \ifx\pdfstringdefDisableCommands\undefined\else
4159 \ifx\pdfstringdefDisableCommands\relax\else

```

```

4160      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4161      \fi
4162      \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.

```

4163 \bbl@trace{Local Language Configuration}
4164 \ifx\loadlocalcfg\undefined
4165   \ifpackagewith{babel}{noconfigs}%
4166     {\let\loadlocalcfg@gobble}%
4167     {\def\loadlocalcfg#1{%
4168       \InputIfFileExists{#1.cfg}%
4169       {\typeout{*****^J%
4170                * Local config file #1.cfg used^J%
4171                *}}%
4172       \@empty}}
4173 \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).

```

4174 \bbl@trace{Language options}
4175 \def\BabelDefinitionFile#1#2#3{}
4176 \let\bbl@afterlang\relax
4177 \let\BabelModifiers\relax
4178 \let\bbl@loaded\@empty
4179 \def\bbl@load@language#1{%
4180   \InputIfFileExists{#1.ldf}%
4181   {\edef\bbl@loaded{\CurrentOption
4182     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4183     \expandafter\let\expandafter\bbl@afterlang
4184       \csname\CurrentOption.ldf-h@k\endcsname
4185     \expandafter\let\expandafter\BabelModifiers
4186       \csname bbl@mod@\CurrentOption\endcsname
4187     \bbl@exp{\AtBeginDocument{%
4188       \bbl@usehooks@lang{\CurrentOption}{\begin{document}}{\CurrentOption}}}%
4189     {\bbl@error{unknown-package-option}}}}

```

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

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

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

```

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@bcplookup\bbl@metalang-\@empty-\@empty-\@empty@@
4197 \ifx\bbl@bcp\relax
4198 \ifx\bbl@opt@main\@nnil
4199 \bbl@error{no-locale-for-meta}{\bbl@metalang}{\}%
4200 \fi
4201 \else
4202 \bbl@read@ini{\bbl@bcp}\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 This will be the main language except if\%
4209 explicitly overridden with 'main='\%
4210 Reported}%
4211 \fi
4212 \endgroup
4213 \fi
4214 \fi
4215 \ifx\bbl@opt@config\@nnil
4216 \ifpackagewith{babel}{noconfigs}{\%
4217 {\InputIfFileExists{bblopts.cfg}%
4218 {\bbl@info{Configuration files are deprecated, as\%
4219 they can break document portability.\%
4220 Reported}%
4221 \typeout{*****^J%
4222 * Local config file bblopts.cfg used^^J%
4223 *}}}%
4224 }{\}%
4225 \else
4226 \InputIfFileExists{\bbl@opt@config.cfg}%
4227 {\bbl@info{Configuration files are deprecated, as\%
4228 they can break document portability.\%
4229 Reported}%
4230 \typeout{*****^J%
4231 * Local config file \bbl@opt@config.cfg used^^J%
4232 *}}}%
4233 {\bbl@error{config-not-found}{\}{\}}{\}%
4234 \fi

```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in `\bbl@language@opts` are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the available locales, and which version (ldf or ini) will be loaded. This is done by first loading the corresponding `babel-(name).tex` file.

The second argument of `\BabelBeforeIni` may contain a `\BabelDefinitionFile` which defines `\bbl@tempa` and `\bbl@tempb` and saves the third argument for the moment of the actual loading. If there is no `\BabelDefinitionFile` the last element is usually empty, and the ini file is loaded. The values are used to build a list in the form 'main-or-not' / 'ldf-or-ldfini-flag' // 'option-name' // 'bcp-tag' / 'ldf-name-or-none'. The 'main-or-not' element is 0 by default and set to 10 later if necessary (by prepending 1). The 'bcp-tag' is stored here so that the corresponding ini file can be loaded directly (with `@import`).

```

4235 \def\BabelBeforeIni#1#2{%
4236 \def\bbl@tempa{\@m}% <- Default if no \BDefFile
4237 \let\bbl@tempb\@empty
4238 #2%
4239 \edef\bbl@toload{%
4240 \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4241 \bbl@toload@last}%
4242 \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4243 \def\BabelDefinitionFile#1#2#3{%
4244 \def\bbl@tempa{#1}\def\bbl@tempb{#2}%

```

```

4245 \namedef{bbl@preldf@CurrentOption}{#3}%
4246 \endinput}%

```

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

```

4247 \def\bbl@tempf{,}
4248 \bbl@foreach\@raw@classoptionslist{%
4249   \in@{=}{#1}%
4250   \ifin@else
4251     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4252   \fi}

```

Store the class/package options in a list. If there is an explicit main, it's placed as the last option. Then loop it to read the tex files, which can have a \BabelDefinitionFile. If there is no tex file, we attempt loading the ldf for the option name; if it fails, an error is raised. Note the option name is surrounded by //...//. Class and package options are separated with @, because errors and info are dealt with in different ways. Consecutive identical languages count as one.

```

4253 \let\bbl@toload\@empty
4254 \let\bbl@toload@last\@empty
4255 \let\bbl@unkopt\@gobble %% <- Ugly
4256 \edef\bbl@tempc{%
4257   \bbl@tempf,@,\bbl@language@opts
4258   \ifx\bbl@opt@main\@nnil\else,\bbl@opt@main\fi}
4259 %
4260 \bbl@foreach\bbl@tempc{%
4261   \in@{@@}{#1}% <- Ugly
4262   \ifin@
4263     \def\bbl@unkopt##1{%
4264       \DeclareOption{##1}{\bbl@error{unknown-package-option}{#1}}}%
4265   \else
4266     \def\CurrentOption{#1}%
4267     \bbl@xin@{//#1//}{\bbl@toload@last}% Collapse consecutive
4268     \ifin@else
4269       \lowercase{\InputIfFileExists{babel-#1.tex}}{#1}%
4270       \IfFileExists{#1.ldf}%
4271       {\edef\bbl@toload{%
4272         \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4273         \bbl@toload@last}%
4274         \edef\bbl@toload@last{0/0//\CurrentOption//und/#1}}%
4275       {\bbl@unkopt{#1}}}%
4276   \fi
4277 \fi}

```

We have to determine (1) if no language has been loaded (in which case we fallback to 'nil', with a special tag), and (2) the main language. With an explicit 'main' language, remove repeated elements. The number 1 flags it as the main language (relevant in *ini* locales), because with 0 becomes 10.

```

4278 \ifx\bbl@opt@main\@nnil
4279   \ifx\bbl@toload@last\@empty
4280     \def\bbl@toload@last{0/0//nil//und-x-nil/nil}
4281     \bbl@info{%
4282       You haven't specified a language as a class or package\\%
4283       option. I'll load 'nil'. Reported}
4284   \fi
4285 \else
4286   \let\bbl@tempc\@empty
4287   \bbl@foreach\bbl@toload{%
4288     \bbl@xin@{//#1//\bbl@opt@main//}{#1}%
4289     \ifin@else
4290       \bbl@add@list\bbl@tempc{#1}%
4291     \fi}
4292   \let\bbl@toload\bbl@tempc
4293 \fi
4294 \edef\bbl@toload{\bbl@toload,1\bbl@toload@last}

```

Finally, load the ‘ini’ file or the pair ‘ini’/‘ldf’ file. Babel resorts to its own mechanism, not the default one based on \ProcessOptions (which is still present to make some internal clean-up). First, handle provide= and friends (with a recursive call if they are present), and then provide=* and friend. \count@ is used as flag: 0 if ‘ini’, 1 if ‘ldf’.

```

4295 \def\AfterBabelLanguage#1{%
4296   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4297 \NewHook{babel/presets}
4298 \UseHook{babel/presets}
4299 %
4300 \let\bbl@tempb\empty
4301 \def\bbl@tempc#1/#2//#3//#4/#5\@@{%
4302   \count@\z@
4303   \ifnum#2=\@m % if no \BabelDefinitionFile
4304     \ifnum#1=\z@ % not main. -- % if provide+=!, provide*=!
4305       \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0//#3//#4/#3\@@
4306       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4307       \fi
4308     \else % 10 = main -- % if provide+=!, provide*=!
4309       \ifodd\bbl@ldfflag\bbl@tempc 10/0//#3//#4/#3\@@
4310       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4311       \fi
4312     \fi
4313   \else
4314     \ifnum#1=\z@ % not main
4315       \ifnum\bbl@iniflag>\@ne\else % if 0, provide
4316         \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4317       \fi
4318     \else % 10 = main
4319       \ifodd\bbl@iniflag\else % if provide+, provide*
4320         \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4321       \fi
4322     \fi
4323     \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4324   \fi}

```

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

```

4325 \def\bbl@tempd#1#2#3#4#5{%
4326   \DeclareOption{#3}{%
4327     \ifcase\count@
4328       \bbl@exp{\bbl@add\bbl@tempb{%
4329         \@nameuse{\bbl@preini#3}%
4330         \bbl@ldfinit
4331         \def\CurrentOption{#3}%
4332         \bbl@babelprovide[\@import=#4,\ifnum#1=\z@\else\bbl@opt@provide,main\fi]{#3}%
4333         \bbl@afterldf}}%
4334     \else
4335       \bbl@add\bbl@tempb{%
4336         \def\CurrentOption{#3}%
4337         \let\localename\CurrentOption
4338         \let\language\localename
4339         \def\BabelIniTag{#4}%
4340         \@nameuse{\bbl@preldf#3}%
4341         \begingroup
4342           \bbl@id@assign
4343           \bbl@read@ini{\BabelIniTag}0%
4344         \endgroup
4345         \bbl@load@language{#5}}%
4346     \fi}
4347 %
4348 \bbl@foreach\bbl@toload{\bbl@tempc#1\@@}
4349 \bbl@tempb
4350 \DeclareOption*{}

```



```

4351 \ProcessOptions
4352 %
4353 \bbl@exp{%
4354   \\AtBeginDocument{\\bbl@usehooks@lang{/}{\begin{document}}{}}}%
4355 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}}{}
4356 </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`

```

4357 <{*kernel>
4358 \let\bbl@onlyswitch\@empty
4359 \input babel.def
4360 \let\bbl@onlyswitch\@undefined
4361 </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.

```

4362 <{*errors>
4363 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4364 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4365 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4366 \catcode`\@=11 \catcode`\^=7
4367 %
4368 \ifx\MessageBreak\@undefined
4369   \gdef\bbl@error@i#1#2{%
4370     \begin{group}
4371       \newlinechar=`^^J
4372       \def\{^^J(babel) }%
4373       \errhelp{#2}\errmessage{\{#1}%
4374     \end{group}}
4375 \else
4376   \gdef\bbl@error@i#1#2{%
4377     \begin{group}
4378       \def\{\MessageBreak}%
4379       \PackageError{babel}{#1}{#2}%
4380     \end{group}}
4381 \fi
4382 \def\bbl@errmessage#1#2#3{%
4383   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4384     \bbl@error@i{#2}{#3}}
4385 % Implicit #2#3#4:
4386 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4387 %
4388 \bbl@errmessage{not-yet-available}
4389   {Not yet available}%
4390   {Find an armchair, sit down and wait}

```

```

4391 \bbl@errmessage{bad-package-option}%
4392 {Bad option '#1=#2'. Either you have misspelled the\\%
4393 key or there is a previous setting of '#1'. Valid\\%
4394 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4395 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4396 {See the manual for further details.}
4397 \bbl@errmessage{base-on-the-fly}
4398 {For a language to be defined on the fly 'base'\\%
4399 is not enough, and the whole package must be\\%
4400 loaded. Either delete the 'base' option or\\%
4401 request the languages explicitly}%
4402 {See the manual for further details.}
4403 \bbl@errmessage{undefined-language}
4404 {You haven't defined the language '#1' yet.\\%
4405 Perhaps you misspelled it or your installation\\%
4406 is not complete}%
4407 {Your command will be ignored, type <return> to proceed}
4408 \bbl@errmessage{invalid-ini-name}
4409 {'#1' not valid with the 'ini' mechanism.\\%
4410 I think you want '#2' instead. You may continue,\\%
4411 but you should fix the name. See the babel manual\\%
4412 for the available locales with 'provide'}%
4413 {See the manual for further details.}
4414 \bbl@errmessage{shorthand-is-off}
4415 {I can't declare a shorthand turned off (\string#2)}
4416 {Sorry, but you can't use shorthands which have been\\%
4417 turned off in the package options}
4418 \bbl@errmessage{not-a-shorthand}
4419 {The character '\string #1' should be made a shorthand character;\\%
4420 add the command \string\usesshorthands\string{#1\string} to
4421 the preamble.\\%
4422 I will ignore your instruction}%
4423 {You may proceed, but expect unexpected results}
4424 \bbl@errmessage{not-a-shorthand-b}
4425 {I can't switch '\string#2' on or off--not a shorthand\\%
4426 This character is not a shorthand. Maybe you made\\%
4427 a typing mistake?}%
4428 {I will ignore your instruction.}
4429 \bbl@errmessage{unknown-attribute}
4430 {The attribute #2 is unknown for language #1.}%
4431 {Your command will be ignored, type <return> to proceed}
4432 \bbl@errmessage{missing-group}
4433 {Missing group for string \string#1}%
4434 {You must assign strings to some category, typically\\%
4435 captions or extras, but you set none}
4436 \bbl@errmessage{only-lua-xe}
4437 {This macro is available only in LuaLaTeX and XeLaTeX.}%
4438 {Consider switching to these engines.}
4439 \bbl@errmessage{only-lua}
4440 {This macro is available only in LuaLaTeX}%
4441 {Consider switching to that engine.}
4442 \bbl@errmessage{unknown-provide-key}
4443 {Unknown key '#1' in \string\babelprovide}%
4444 {See the manual for valid keys}%
4445 \bbl@errmessage{unknown-mapfont}
4446 {Option '\bbl@KVP@mapfont' unknown for\\%
4447 mapfont. Use 'direction'}%
4448 {See the manual for details.}
4449 \bbl@errmessage{no-ini-file}
4450 {There is no ini file for the requested language\\%
4451 (#1: \language). Perhaps you misspelled it or your\\%
4452 installation is not complete}%
4453 {Fix the name or reinstall babel.}

```

```

4454 \bbl@errmessage{digits-is-reserved}
4455 {The counter name 'digits' is reserved for mapping\\%
4456 decimal digits}%
4457 {Use another name.}
4458 \bbl@errmessage{limit-two-digits}
4459 {Currently two-digit years are restricted to the\\
4460 range 0-9999}%
4461 {There is little you can do. Sorry.}
4462 \bbl@errmessage{alphabetic-too-large}
4463 {Alphabetic numeral too large (#1)}%
4464 {Currently this is the limit.}
4465 \bbl@errmessage{no-ini-info}
4466 {I've found no info for the current locale.\\%
4467 The corresponding ini file has not been loaded\\%
4468 Perhaps it doesn't exist}%
4469 {See the manual for details.}
4470 \bbl@errmessage{unknown-ini-field}
4471 {Unknown field '#1' in \string\BCPdata.\\%
4472 Perhaps you misspelled it}%
4473 {See the manual for details.}
4474 \bbl@errmessage{unknown-locale-key}
4475 {Unknown key for locale '#2':\\%
4476 #3\\%
4477 \string#1 will be set to \string\relax}%
4478 {Perhaps you misspelled it.}%
4479 \bbl@errmessage{adjust-only-vertical}
4480 {Currently, #1 related features can be adjusted only\\%
4481 in the main vertical list}%
4482 {Maybe things change in the future, but this is what it is.}
4483 \bbl@errmessage{layout-only-vertical}
4484 {Currently, layout related features can be adjusted only\\%
4485 in vertical mode}%
4486 {Maybe things change in the future, but this is what it is.}
4487 \bbl@errmessage{bidi-only-lua}
4488 {The bidi method 'basic' is available only in\\%
4489 luatex. I'll continue with 'bidi=default', so\\%
4490 expect wrong results.\\%
4491 Suggested actions:\\%
4492 * If possible, switch to luatex, as xetex is not\\%
4493 recommend anymore.\\
4494 * If you can't, try 'bidi=bidi' with xetex.\\%
4495 * With pdftex, only 'bidi=default' is available.}%
4496 {See the manual for further details.}
4497 \bbl@errmessage{multiple-bidi}
4498 {Multiple bidi settings inside a group\\%
4499 I'll insert a new group, but expect wrong results.\\%
4500 Suggested action:\\%
4501 * Add a new group where appropriate.}
4502 {See the manual for further details.}
4503 \bbl@errmessage{unknown-package-option}
4504 {Unknown option '\CurrentOption'.\\%
4505 Suggested actions:\\%
4506 * Make sure you haven't misspelled it\\%
4507 * Check in the babel manual that it's supported\\%
4508 * If supported and it's a language, you may\\%
4509 \space\space need in some distributions a separate\\%
4510 \space\space installation\\%
4511 * If installed, check there isn't an old\\%
4512 \space\space version of the required files in your system}
4513 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4514 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4515 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4516 \bbl@errmessage{config-not-found}

```

```

4517 {Local config file '\bbl@opt@config.cfg' not found.\\%
4518   Suggested actions:\\%
4519   * Make sure you haven't misspelled it in config=\\%
4520   * Check it exists and it's in the correct path}%
4521 {Perhaps you misspelled it.}
4522 \bbl@errmessage{late-after-babel}
4523 {Too late for \string\AfterBabelLanguage}%
4524 {Languages have been loaded, so I can do nothing}
4525 \bbl@errmessage{double-hyphens-class}
4526 {Double hyphens aren't allowed in \string\babelcharclass\\%
4527   because it's potentially ambiguous}%
4528 {See the manual for further info}
4529 \bbl@errmessage{unknown-interchar}
4530 {'#1' for '\language' cannot be enabled.\\%
4531   Maybe there is a typo}%
4532 {See the manual for further details.}
4533 \bbl@errmessage{unknown-interchar-b}
4534 {'#1' for '\language' cannot be disabled.\\%
4535   Maybe there is a typo}%
4536 {See the manual for further details.}
4537 \bbl@errmessage{charproperty-only-vertical}
4538 {\string\babelcharproperty\space can be used only in\\%
4539   vertical mode (preamble or between paragraphs)}%
4540 {See the manual for further info}
4541 \bbl@errmessage{unknown-char-property}
4542 {No property named '#2'. Allowed values are\\%
4543   direction (bc), mirror (bmg), and linebreak (lb)}%
4544 {See the manual for further info}
4545 \bbl@errmessage{bad-transform-option}
4546 {Bad option '#1' in a transform.\\%
4547   I'll ignore it but expect more errors}%
4548 {See the manual for further info.}
4549 \bbl@errmessage{font-conflict-transforms}
4550 {Transforms cannot be re-assigned to different\\%
4551   fonts. The conflict is in '\bbl@kv@label'.\\%
4552   Apply the same fonts or use a different label}%
4553 {See the manual for further details.}
4554 \bbl@errmessage{transform-not-available}
4555 {'#1' for '\language' cannot be enabled.\\%
4556   Maybe there is a typo or it's a font-dependent transform}%
4557 {See the manual for further details.}
4558 \bbl@errmessage{transform-not-available-b}
4559 {'#1' for '\language' cannot be disabled.\\%
4560   Maybe there is a typo or it's a font-dependent transform}%
4561 {See the manual for further details.}
4562 \bbl@errmessage{year-out-range}
4563 {Year out of range.\\%
4564   The allowed range is #1}%
4565 {See the manual for further details.}
4566 \bbl@errmessage{only-pdftex-lang}
4567 {The '#1' ldf style doesn't work with #2,\\%
4568   but you can use the ini locale instead.\\%
4569   Try adding 'provide=*' to the option list. You may\\%
4570   also want to set 'bidi=' to some value}%
4571 {See the manual for further details.}
4572 \bbl@errmessage{hyphenmins-args}
4573 {\string\babelhyphenmins\ accepts either the optional\\%
4574   argument or the star, but not both at the same time}%
4575 {See the manual for further details.}
4576 \bbl@errmessage{no-locale-for-meta}
4577 {There isn't currently a locale for the 'lang' requested\\%
4578   in the PDF metadata ('#1'). To fix it, you can\\%
4579   set explicitly a similar language (using the same\\%

```

```

4580 script) with the key main= when loading babel. If you\\%
4581 continue, I'll fallback to the 'nil' language, with\\%
4582 tag 'und' and script 'Latn', but expect a bad font\\%
4583 rendering with other scripts. You may also need set\\%
4584 explicitly captions and date, too}%
4585 {See the manual for further details.}
4586 </errors>
4587 <*patterns>

```

8. Loading hyphenation patterns

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

```

4588 <@Make sure ProvidesFile is defined@>
4589 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4590 \xdef\bbl@format{\jobname}
4591 \def\bbl@version{<@version@>}
4592 \def\bbl@date{<@date@>}
4593 \ifx\AtBeginDocument\@undefined
4594 \def\@empty{}
4595 \fi
4596 <@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.

```

4597 \def\process@line#1#2 #3 #4 {%
4598 \ifx=#1%
4599 \process@synonym{#2}%
4600 \else
4601 \process@language{#1#2}{#3}{#4}%
4602 \fi
4603 \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.

```

4604 \toks@{}
4605 \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.

```

4606 \def\process@synonym#1{%
4607 \ifnum\last@language=\m@ne
4608 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4609 \else
4610 \expandafter\chardef\csname l@#1\endcsname\last@language
4611 \wlog{\string\l@#1=\string\language\the\last@language}%
4612 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4613 \csname\language\hyphenmins\endcsname
4614 \let\bbl@elt\relax
4615 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4616 \fi}

```

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

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

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

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

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

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

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

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

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

```

4617 \def\process@language#1#2#3{%
4618   \expandafter\addlanguage\csname l@#1\endcsname
4619   \expandafter\language\csname l@#1\endcsname
4620   \edef\language#1#2#3{%
4621     \bbl@hook@everylanguage{#1}%
4622     % > luatex
4623     \bbl@get@enc#1::\@@@
4624     \begingroup
4625       \lefthyphenmin\m@ne
4626       \bbl@hook@loadpatterns{#2}%
4627       % > luatex
4628       \ifnum\lefthyphenmin=\m@ne
4629       \else
4630         \expandafter\xdef\csname #1hyphenmins\endcsname{%
4631           \the\lefthyphenmin\the\righthyphenmin}%
4632       \fi
4633     \endgroup
4634     \def\bbl@tempa{#3}%
4635     \ifx\bbl@tempa\@empty\else
4636       \bbl@hook@loadexceptions{#3}%
4637       % > luatex
4638     \fi
4639     \let\bbl@elt\relax
4640     \edef\bbl@languages{%
4641       \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4642     \ifnum\the\language=\z@
4643       \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4644         \set@hyphenmins\tw@thr@@\relax
4645       \else
4646         \expandafter\expandafter\expandafter\set@hyphenmins
4647         \csname #1hyphenmins\endcsname
4648       \fi
4649       \the\toks@
4650       \toks@{}%
4651     \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.

```
4652 \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.

```
4653 \def\bbl@hook@everylanguage#1{}
4654 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4655 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4656 \def\bbl@hook@loadkernel#1{%
4657   \def\addlanguage{\csname newlanguage\endcsname}%
4658   \def\adddialect##1##2{%
4659     \global\chardef##1##2\relax
4660     \wlog{\string##1 = a dialect from \string\language##2}}%
4661   \def\iflanguage##1{%
4662     \expandafter\ifx\csname l@##1\endcsname\relax
4663       \nolanner{##1}%
4664     \else
4665       \ifnum\csname l@##1\endcsname=\language
4666         \expandafter\expandafter\expandafter\@firstoftwo
4667       \else
4668         \expandafter\expandafter\expandafter\@secondoftwo
4669       \fi
4670     \fi}%
4671   \def\providehyphenmins##1##2{%
4672     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4673       \@namedef{##1hyphenmins}{##2}%
4674     \fi}%
4675   \def\set@hyphenmins##1##2{%
4676     \lefthyphenmin##1\relax
4677     \righthyphenmin##2\relax}%
4678   \def\selectlanguage{%
4679     \errhelp{Selecting a language requires a package supporting it}%
4680     \errmessage{No multilingual package has been loaded}}%
4681   \let\foreignlanguage\selectlanguage
4682   \let\otherlanguage\selectlanguage
4683   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4684   \def\bbl@usehooks##1##2{%
4685     \def\setlocale{%
4686       \errhelp{Find an armchair, sit down and wait}%
4687       \errmessage{(babel) Not yet available}}%
4688     \let\uselocale\setlocale
4689     \let\locale\setlocale
4690     \let\selectlocale\setlocale
4691     \let\localename\setlocale
4692     \let\textlocale\setlocale
4693     \let\textlanguage\setlocale
4694     \let\languagetext\setlocale}
4695   \begingroup
4696     \def\AddBabelHook#1#2{%
4697       \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4698         \def\next{\toks1}%
4699       \else
4700         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4701       \fi
4702       \next}
4703   \ifx\directlua\@undefined
4704     \ifx\XeTeXinputencoding\@undefined\else
4705       \input xebabel.def
4706     \fi
4707   \else
4708     \input luababel.def
4709   \fi
4710   \openin1 = babel-\bbl@format.cfg
```

```

4711 \ifeof1
4712 \else
4713   \input babel-\bbl@format.cfg\relax
4714 \fi
4715 \closein1
4716 \endgroup
4717 \bbl@hook@loadkernel{switch.def}

```

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

```

4718 \openin1 = language.dat

```

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

```

4719 \def\language#1{english}%
4720 \ifeof1
4721 \message{I couldn't find the file language.dat,\space
4722          I will try the file hyphen.tex}
4723 \input hyphen.tex\relax
4724 \chardef\l@english\z@
4725 \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`.

```

4726 \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.

```

4727 \loop
4728   \endlinechar@m@ne
4729   \read1 to \bbl@line
4730   \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.

```

4731   \if T\ifeof1F\fi T\relax
4732   \ifx\bbl@line\@empty\else
4733     \edef\bbl@line{\bbl@line\space\space\space}%
4734     \expandafter\process@line\bbl@line\relax
4735   \fi
4736 \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.

```

4737 \begingroup
4738 \def\bbl@elt#1#2#3#4{%
4739   \global\language=#2\relax
4740   \gdef\language#1{%
4741     \def\bbl@elt##1##2##3##4{}}%
4742 \bbl@languages
4743 \endgroup
4744 \fi
4745 \closein1

```

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

```

4746 \if/\the\toks@\else
4747 \errhelp{language.dat loads no language, only synonyms}
4748 \errmessage{Orphan language synonym}
4749 \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.

```

4750 \let\bbl@line\@undefined
4751 \let\process@line\@undefined
4752 \let\process@synonym\@undefined
4753 \let\process@language\@undefined
4754 \let\bbl@get@enc\@undefined
4755 \let\bbl@hyph@enc\@undefined
4756 \let\bbl@tempa\@undefined
4757 \let\bbl@hook@loadkernel\@undefined
4758 \let\bbl@hook@everylanguage\@undefined
4759 \let\bbl@hook@loadpatterns\@undefined
4760 \let\bbl@hook@loadexceptions\@undefined
4761 </patterns>

```

Here the code for iniTeX ends.

9. luatex + xetex: common stuff

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

```

4762 <<*More package options>> ≡
4763 \chardef\bbl@bidimode\z@
4764 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4765 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4766 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4767 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4768 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4769 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4770 <</More package options>>

```

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

```

4771 <<*Font selection>> ≡
4772 \bbl@trace{Font handling with fontspec}
4773 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4774 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4775 \DisableBabelHook{babel-fontspec}
4776 \@onlypreamble\babelfont
4777 \ifx\NewDocumentCommand\@undefined\else % Not plain
4778   \NewDocumentCommand\babelfont{0}{m0}{m0}}{%
4779     \bbl@bblfont@o[#1]{#2}{#3,#5}{#4}}
4780 \fi
4781 \newcommand\bbl@bblfont@o[2][]{% 1=langs/scripts 2=fam
4782   \ifx\fontspec\@undefined
4783     \usepackage{fontspec}%
4784   \fi
4785   \EnableBabelHook{babel-fontspec}%
4786   \edef\bbl@tempa{#1}%
4787   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4788   \bbl@bblfont}
4789 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4790   \bbl@ifunset{\bbl@tempb family}%
4791     {\bbl@providfam{\bbl@tempb}}%
4792   }%
4793   % For the default font, just in case:
4794   \bbl@ifunset{\bbl@lsys@\language\name}{\bbl@provide@lsys{\language\name}}{%
4795     \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4796     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@

```

```

4797 \bbl@exp{%
4798 \let\<bbl@bbl@tempb dflt@>\language\<bbl@bbl@tempb dflt@>%
4799 \\\bbl@font@set\<bbl@bbl@tempb dflt@>\language\<bbl@bbl@tempb dflt@>%
4800 \<bbl@tempb default>\<bbl@tempb family>}}%
4801 {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4802 \bbl@csarg\def\bbl@tempb dflt@##1{\<#1>\<#2>}}}%

```

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

```

4803 \def\bbl@providfam#1{%
4804 \bbl@exp{%
4805 \\\newcommand\<#1default>{% Just define it
4806 \\\bbl@add@list\\bbl@font@fams{#1}%
4807 \\\NewHook{#1family}%
4808 \\\DeclareRobustCommand\<#1family>{%
4809 \\\not@math@alphabet\<#1family>\relax
4810 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4811 \\\fontfamily\<#1default>%
4812 \\\UseHook{#1family}%
4813 \\\selectfont}%
4814 \\\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.

```

4815 \def\bbl@nostdfont#1{%
4816 \bbl@once{nostdfam-\f@family}%
4817 {\bbl@infowarn{The current font is not a babel standard family:\%
4818 #1%
4819 \fontname\font\\%
4820 There is nothing intrinsically wrong, and you can\\%,
4821 ignore this message altogether if you do not need\\%
4822 this font. If they are used in the document, be aware\\%
4823 'babel' will not set Script and Language for it, so\\%
4824 you may consider defining a new family with \string\babelfont.\%
4825 See the manual for further details about \string\babelfont.
4826 Reported}}
4827 {}}%
4828 \gdef\bbl@switchfont{%
4829 \bbl@ifunset\bbl@sys@\language\{\bbl@provide@sys@\language\}}}%
4830 \bbl@exp{% e.g., Arabic -> arabic
4831 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4832 \bbl@foreach\bbl@font@fams{%
4833 \bbl@ifunset\bbl@##1dflt@\language}% (1) language?
4834 {\bbl@ifunset\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4835 {\bbl@ifunset\bbl@##1dflt@}% 2=F - (3) from generic?
4836 {}% 123=F - nothing!
4837 {\bbl@exp{% 3=T - from generic
4838 \global\let\<bbl@##1dflt@\language>%
4839 \<bbl@##1dflt@>}}}%
4840 {\bbl@exp{% 2=T - from script
4841 \global\let\<bbl@##1dflt@\language>%
4842 \<bbl@##1dflt@*\bbl@tempa>}}}%
4843 {}% 1=T - language, already defined
4844 \def\bbl@tempa{\bbl@nostdfont}}}%
4845 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4846 \bbl@ifunset\bbl@##1dflt@\language}%
4847 {\bbl@cs{famrst@##1}%
4848 \global\bbl@csarg\let{famrst@##1}\relax}%
4849 {\bbl@exp{% order is relevant.
4850 \\\bbl@add\\originalTeX{%
4851 \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4852 \<##1default>\<##1family>{##1}}%
4853 \\\bbl@font@set\<bbl@##1dflt@\language>% the main part!
4854 \<##1default>\<##1family>}}}%
4855 \bbl@ifrestoring{\bbl@tempa}}%

```

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

```

4856 \ifx\f@family\undefined\else % if latex
4857 \ifcase\bbl@engine % if pdftex
4858 \let\bbl@cckstdfonts\relax
4859 \else
4860 \def\bbl@cckstdfonts{%
4861 \begingroup
4862 \global\let\bbl@cckstdfonts\relax
4863 \let\bbl@tempa\empty
4864 \bbl@foreach\bbl@font@fams{%
4865 \bbl@ifunset\bbl@##1dflt@{%
4866 {\@nameuse{##1family}}%
4867 \bbl@csgarg\gdef{WFF@f@family}}}% Flag
4868 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4869 \space\space\fontname\font\\}%
4870 \bbl@csgarg\xdef{##1dflt@}{\f@family}%
4871 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4872 {}}%
4873 \ifx\bbl@tempa\empty\else
4874 \bbl@infowarn{The following font families will use the default\\%
4875 settings for all or some languages:\\%
4876 \bbl@tempa
4877 There is nothing intrinsically wrong with it, but\\%
4878 'babel' will no set Script and Language, which could\\%
4879 be relevant in some languages. If your document uses\\%
4880 these families, consider redefining them with \string\babelfont.\\%
4881 Reported}%
4882 \fi
4883 \endgroup}
4884 \fi
4885 \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*`).

```

4886 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4887 \bbl@xin@{<}{#1}%
4888 \ifin@
4889 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4890 \fi
4891 \bbl@exp{% 'Unprotected' macros return prev values
4892 \def\\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
4893 \\bbl@ifsamestring{#2}{\f@family}%
4894 {\#3%
4895 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4896 \let\\bbl@tempa\relax}%
4897 {}}}

```

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).

```

4898 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4899 \let\bbl@tempe\bbl@mapselect

```

```

4900 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4901 \bbl@exp{\bbl@replace{\bbl@tempb{\bbl@stripslash\family/}}}%
4902 \let\bbl@mapselect\relax
4903 \let\bbl@tempfam#4% e.g., '\rmfamily', to be restored below
4904 \let#4\@empty % Make sure \renewfontfamily is valid
4905 \bbl@set@renderer
4906 \bbl@exp{%
4907   \let\bbl@tempfam\<\bbl@stripslash#4\space>% e.g., '\rmfamily '
4908   \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4909   {\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4910   \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4911   {\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4912   \renewfontfamily\#4%
4913   [\bbl@cl{lsys},% xetex removes unknown features :-(
4914   \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4915   #2]}{#3}% i.e., \bbl@exp{.}{#3}
4916 \bbl@unset@renderer
4917 \begingroup
4918 #4%
4919 \xdef#1{\f@family}% e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4920 \endgroup
4921 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4922 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4923 \ifin@
4924 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4925 \fi
4926 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4927 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4928 \ifin@
4929 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4930 \fi
4931 \let#4\bbl@tempfam
4932 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@tempfam
4933 \let\bbl@mapselect\bbl@tempe}%

```

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

```

4934 \def\bbl@font@rst#1#2#3#4{%
4935   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4936 \def\bbl@font@fams{rm,sf,tt}
4937 <</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.

```

4938 <*>xetex<
4939 \def\BabelStringsDefault{unicode}
4940 \let\xebbl@stop\relax
4941 \AddBabelHook{xetex}{encodedcommands}{%
4942   \def\bbl@tempa{#1}%
4943   \ifx\bbl@tempa\@empty
4944     \XeTeXinputencoding"bytes"%
4945   \else
4946     \XeTeXinputencoding"#1"%
4947   \fi

```

```

4948 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4949 \AddBabelHook{xetex}{stopcommands}{%
4950 \xebbl@stop
4951 \let\xebbl@stop\relax}
4952 \def\bbl@input@classes{% Used in CJK intraspaces
4953 \input{load-unicode-xetex-classes.tex}%
4954 \let\bbl@input@classes\relax}
4955 \def\bbl@intraspace#1 #2 #3\@@{%
4956 \bbl@csarg\gdef{xeisp@\language}%
4957 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4958 \def\bbl@intrapenalty#1\@@{%
4959 \bbl@csarg\gdef{xeipn@\language}%
4960 {\XeTeXlinebreakpenalty #1\relax}}
4961 \def\bbl@provide@intraspace{%
4962 \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4963 \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4964 \ifin@
4965 \bbl@ifunset{bbl@intsp@\language}{}%
4966 {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4967 \ifx\bbl@KVP@intraspace\@nnil
4968 \bbl@exp{%
4969 \\\bbl@intraspace\bbl@cl{intsp}\@@}%
4970 \fi
4971 \ifx\bbl@KVP@intrapenalty\@nnil
4972 \bbl@intrapenalty0\@@
4973 \fi
4974 \fi
4975 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4976 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4977 \fi
4978 \ifx\bbl@KVP@intrapenalty\@nnil\else
4979 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4980 \fi
4981 \bbl@exp{%
4982 \\\bbl@add<extras\language>{%
4983 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4984 <\bbl@xeisp@\language>%
4985 <\bbl@xeipn@\language>%
4986 \\\bbl@toglobal<extras\language>%
4987 \\\bbl@add<noextras\language>{%
4988 \XeTeXlinebreaklocale ""}%
4989 \\\bbl@toglobal<noextras\language>%
4990 \ifx\bbl@ispacesize\undefined
4991 \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4992 \ifx\AtBeginDocument\@notprerr
4993 \expandafter\@secondoftwo % to execute right now
4994 \fi
4995 \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4996 \fi}%
4997 \fi}
4998 \ifx\DisableBabelHook\@undefined\endinput\fi
4999 \let\bbl@set@renderer\relax
5000 \let\bbl@unset@renderer\relax
5001 <@Font selection@>
5002 \def\bbl@provide@extra#1{}

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

5003 \def\bbl@xenohyph@d{%
5004 \bbl@ifset{bbl@prehc@\language}%
5005 {\ifnum\hyphenchar\font=\defaultthyphenchar
5006 \iffontchar\font\bbl@cl{prehc}\relax
5007 \hyphenchar\font\bbl@cl{prehc}\relax
5008 \else\iffontchar\font"200B

```

```

5009      \hyphenchar\font"200B
5010      \else
5011      \bbl@warning
5012      {Neither 0 nor ZERO WIDTH SPACE are available\\%
5013      in the current font, and therefore the hyphen\\%
5014      will be printed. Try changing the fontspec's\\%
5015      'HyphenChar' to another value, but be aware\\%
5016      this setting is not safe (see the manual).\\%
5017      Reported}%
5018      \hyphenchar\font\defaultthyphenchar
5019      \fi\fi
5020      \fi}%
5021      {\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.

```

5022 \ifnum\Xe@alloc@intercharclass<\thr@@
5023 \Xe@alloc@intercharclass\thr@@
5024 \fi
5025 \chardef\bbl@xe@class@default@=\z@
5026 \chardef\bbl@xe@class@cjkideogram@=\@ne
5027 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
5028 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
5029 \chardef\bbl@xe@class@boundary@=4095
5030 \chardef\bbl@xe@class@ignore@=4096

```

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

```

5031 \AddBabelHook{babel-interchar}{beforeextras}{%
5032 \@nameuse{bbl@xechars@\language@}}
5033 \DisableBabelHook{babel-interchar}
5034 \protected\def\bbl@charclass#1{%
5035 \ifnum\count@<\z@
5036 \count@-\count@
5037 \loop
5038 \bbl@exp{%
5039 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5040 \XeTeXcharclass\count@ \bbl@tempc
5041 \ifnum\count@<`#1\relax
5042 \advance\count@ \@ne
5043 \repeat
5044 \else
5045 \babel@savevariable{\XeTeXcharclass`#1}%
5046 \XeTeXcharclass`#1 \bbl@tempc
5047 \fi
5048 \count@`#1\relax}

```

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

```

5049 \newcommand\bbl@ifinterchar[1]{%
5050 \let\bbl@tempa\@gobble % Assume to ignore
5051 \edef\bbl@tempb{\zap@space#1 \@empty}%
5052 \ifx\bbl@KVP@interchar\@nnil\else
5053 \bbl@replace\bbl@KVP@interchar{ }{,}%
5054 \bbl@foreach\bbl@tempb{%

```

```

5055 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5056 \ifin@
5057 \let\bbl@tempa\@firstofone
5058 \fi}%
5059 \fi
5060 \bbl@tempa}
5061 \newcommand\IfBabelIntercharT[2]{%
5062 \bbl@carg\bbl@add{\bbl@icsave@\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5063 \newcommand\babelcharclass[3]{%
5064 \EnableBabelHook{babel-interchar}%
5065 \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5066 \def\bbl@tempb##1{%
5067 \ifx##1\@empty\else
5068 \ifx##1-%
5069 \bbl@upto
5070 \else
5071 \bbl@charclass{%
5072 \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5073 \fi
5074 \expandafter\bbl@tempb
5075 \fi}%
5076 \bbl@ifunset{\bbl@xechars@#1}%
5077 {\toks@{%
5078 \babel@savevariableXeTeXinterchartokenstate
5079 XeTeXinterchartokenstate\@ne
5080 }}%
5081 {\toks@\expandafter\expandafter\expandafter{%
5082 \csname bbl@xechars@#1\endcsname}}%
5083 \bbl@csarg\edef{xechars@#1}{%
5084 \the\toks@
5085 \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5086 \bbl@tempb#3\@empty}}
5087 \protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
5088 \protected\def\bbl@upto{%
5089 \ifnum\count@>\z@
5090 \advance\count@\@ne
5091 \count@-\count@
5092 \else\ifnum\count@=\z@
5093 \bbl@charclass{-}%
5094 \else
5095 \bbl@error{double-hyphens-class}{\count@}{\count@}%
5096 \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>`.

```

5097 \def\bbl@ignoreinterchar{%
5098 \ifnum\language=\l@nohyphenation
5099 \expandafter\@gobble
5100 \else
5101 \expandafter\@firstofone
5102 \fi}
5103 \newcommand\babelinterchar[5][{}]{%
5104 \let\bbl@kv@label\@empty
5105 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5106 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5107 {\bbl@ignoreinterchar{#5}}%
5108 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5109 \bbl@expf{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5110 \bbl@expf{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5111 XeTeXinterchartoks
5112 \@nameuse{\bbl@xeclass@\bbl@tempa @#2}}{#2}} %
5113 \bbl@ifunset{\bbl@xeclass@\bbl@tempa @#2}{#2}} %

```

```

5114 \nameuse{bbl@xeclasse@bbl@tempb @%
5115 \bbl@ifunset{bbl@xeclasse@bbl@tempb @#2}{#2}} %
5116 = \expandafter{%
5117 \csname bbl@ic@bbl@kv@label @#2\expandafter\endcsname
5118 \csname\zap@space bbl@xeinter@bbl@kv@label
5119 @#3@#4@#2 \@empty\endcsname}}}}
5120 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5121 \bbl@ifunset{bbl@ic@#1@language}%
5122 {\bbl@error{unknown-interchar}{#1}{}}}%
5123 {\bbl@csarg\let{ic@#1@language}\@firstofone}}
5124 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5125 \bbl@ifunset{bbl@ic@#1@language}%
5126 {\bbl@error{unknown-interchar-b}{#1}{}}}%
5127 {\bbl@csarg\let{ic@#1@language}\@gobble}}
5128 </xetex

```

10.3. Layout

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

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

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

```

5129 < *xetex | texxet
5130 \providecommand\bbl@provide@intraspace{}
5131 \bbl@trace{Redefinitions for bidi layout}

Finish here if there is no layout.

5132 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5133 \IfBabelLayout{nopars}
5134 {}
5135 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5136 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5137 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5138 \ifnum\bbl@bidimode>\z@
5139 \IfBabelLayout{pars}
5140 {\def\hangfrom#1{%
5141 \setbox\@tempboxa\hbox{#1}}%
5142 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5143 \noindent\box\@tempboxa}
5144 \def\raggedright{%
5145 \let\@centercr
5146 \bbl@startskip\z@skip
5147 \@rightskip\@flushglue
5148 \bbl@endskip\@rightskip
5149 \parindent\z@
5150 \parfillskip\bbl@startskip}
5151 \def\raggedleft{%
5152 \let\@centercr
5153 \bbl@startskip\@flushglue
5154 \bbl@endskip\z@skip
5155 \parindent\z@
5156 \parfillskip\bbl@endskip}}
5157 {}
5158 \fi
5159 \IfBabelLayout{lists}
5160 {\bbl@sreplace\list
5161 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5162 \def\bbl@listleftmargin{%
5163 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5164 \ifcase\bbl@engine

```



```

5165 \def\labelenumii{}\theenumii()% pdfTeX doesn't reverse ()
5166 \def\p@enumiii{\p@enumii}\theenumii()%
5167 \fi
5168 \bbl@sreplace\@verbatim
5169 {\leftskip\@totalleftmargin}%
5170 {\bbl@startskip\textwidth
5171 \advance\bbl@startskip-\linewidth}%
5172 \bbl@sreplace\@verbatim
5173 {\rightskip\z@skip}%
5174 {\bbl@endskip\z@skip}}%
5175 {}
5176 \IfBabelLayout{contents}
5177 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5178 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5179 {}
5180 \IfBabelLayout{columns}
5181 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5182 \def\bbl@outputbox#1{%
5183 \hb@xt@\textwidth{%
5184 \hskip\columnwidth
5185 \hfil
5186 {\normalcolor\vrule \@width\columnseprule}%
5187 \hfil
5188 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5189 \hskip-\textwidth
5190 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5191 \hskip\columnsep
5192 \hskip\columnwidth}}}%
5193 {}

```

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

```

5194 \IfBabelLayout{counters*}%
5195 {\bbl@add\bbl@opt@layout{.counters.}%
5196 \AddToHook{shipout/before}{%
5197 \let\bbl@tempa\babelsublr
5198 \let\babelsublr\@firstofone
5199 \let\bbl@save@thepage\thepage
5200 \protected@edef\thepage{\thepage}%
5201 \let\babelsublr\bbl@tempa}%
5202 \AddToHook{shipout/after}{%
5203 \let\thepage\bbl@save@thepage}}{}
5204 \IfBabelLayout{counters}%
5205 {\let\bbl@latinarabic=\@arabic
5206 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5207 \let\bbl@asciroman=\@roman
5208 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5209 \let\bbl@asciiRoman=\@Roman
5210 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5211 \fi % end if layout
5212 </xetex | texxet

```

10.4. 8-bit TeX

Which start just above, because some code is shared with `xetex`. Now, 8-bit specific stuff. If just one encoding has been declared, then assume no switching is necessary (1).

```

5213 < *texxet
5214 \def\bbl@provide@extra#1{%
5215 % == auto-select encoding ==
5216 \ifx\bbl@encoding@select\off\@empty\else
5217 \bbl@ifunset{\bbl@encoding@#1}%
5218 {\def\@elt##1{,##1,}%
5219 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%

```

```

5220 \count@z@
5221 \bbl@foreach\bbl@tempe{%
5222   \def\bbl@tempd{##1}% Save last declared
5223   \advance\count@\@ne}%
5224   \ifnum\count@>\@ne % (1)
5225     \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5226     \ifx\bbl@tempa\relax \let\bbl@tempa\empty \fi
5227     \bbl@replace\bbl@tempa{ },}%
5228     \global\bbl@csarg\let{encoding@#1}\empty
5229     \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5230     \ifin@else % if main encoding included in ini, do nothing
5231       \let\bbl@tempb\relax
5232       \bbl@foreach\bbl@tempa{%
5233         \ifx\bbl@tempb\relax
5234           \bbl@xin@{,##1,}{,\bbl@tempe,}%
5235           \ifin@\def\bbl@tempb{##1}\fi
5236         \fi}%
5237       \ifx\bbl@tempb\relax\else
5238         \bbl@exp{%
5239           \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5240           \gdef\<bbl@encoding@#1>{%
5241             \\babel@save\\f@encoding
5242             \\bbl@add\\originalTeX{\\selectfont}%
5243             \\fontencoding{\bbl@tempb}%
5244             \\selectfont}}%
5245         \fi
5246       \fi
5247     \fi}%
5248   }%
5249 \fi}
5250 </texet>

```

10.5. LuaTeX

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

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

```

5251 *luatex[]
5252 \directlua{ Babel = Babel or {} } % DL2
5253 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5254 \bbl@trace{Read language.dat}
5255 \ifx\bbl@readstream\undefined
5256 \csname newread\endcsname\bbl@readstream
5257 \fi
5258 \begingroup
5259 \toks@{}
5260 \count@ \z@ % 0=start, 1=0th, 2=normal
5261 \def\bbl@process@line#1#2 #3 #4 {%
5262 \ifx=#1%
5263 \bbl@process@synonym{#2}%
5264 \else
5265 \bbl@process@language{#1#2}{#3}{#4}%
5266 \fi
5267 \ignorespaces}
5268 \def\bbl@manylang{%
5269 \ifnum\bbl@last>\@ne
5270 \bbl@info{Non-standard hyphenation setup}%
5271 \fi
5272 \let\bbl@manylang\relax}
5273 \def\bbl@process@language#1#2#3{%
5274 \ifcase\count@
5275 \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5276 \or
5277 \count@\tw@
5278 \fi
5279 \ifnum\count@=\tw@
5280 \expandafter\addlanguage\csname l@#1\endcsname
5281 \language\allocationnumber
5282 \chardef\bbl@last\allocationnumber
5283 \bbl@manylang
5284 \let\bbl@elt\relax
5285 \xdef\bbl@languages{%
5286 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5287 \fi
5288 \the\toks@
5289 \toks@{}}
5290 \def\bbl@process@synonym@aux#1#2{%
5291 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5292 \let\bbl@elt\relax
5293 \xdef\bbl@languages{%
5294 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5295 \def\bbl@process@synonym#1{%
5296 \ifcase\count@
5297 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5298 \or
5299 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{%
5300 \else
5301 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5302 \fi}
5303 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5304 \chardef\l@english\z@
5305 \chardef\l@USenglish\z@
5306 \chardef\bbl@last\z@
5307 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
5308 \gdef\bbl@languages{%
5309 \bbl@elt{english}{0}{\hyphen.tex}}%
5310 \bbl@elt{USenglish}{0}{}}

```

```

5311 \else
5312 \global\let\bbl@languages@format\bbl@languages
5313 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5314 \ifnum#2>\z@else
5315 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5316 \fi}%
5317 \xdef\bbl@languages{\bbl@languages}%
5318 \fi
5319 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
5320 \bbl@languages
5321 \openin\bbl@readstream=language.dat
5322 \ifeof\bbl@readstream
5323 \bbl@warning{I couldn't find language.dat. No additional\\%
5324 patterns loaded. Reported}%
5325 \else
5326 \loop
5327 \endlinechar\m@ne
5328 \read\bbl@readstream to \bbl@line
5329 \endlinechar\^^M
5330 \if T\ifeof\bbl@readstream F\fi T\relax
5331 \ifx\bbl@line\@empty\else
5332 \edef\bbl@line{\bbl@line\space\space\space}%
5333 \expandafter\bbl@process@line\bbl@line\relax
5334 \fi
5335 \repeat
5336 \fi
5337 \closein\bbl@readstream
5338 \endgroup
5339 \bbl@trace{Macros for reading patterns files}
5340 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
5341 \ifx\babelcatcodetablenum\@undefined
5342 \ifx\newcatcodetable\@undefined
5343 \def\babelcatcodetablenum{5211}
5344 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5345 \else
5346 \newcatcodetable\babelcatcodetablenum
5347 \newcatcodetable\bbl@pattcodes
5348 \fi
5349 \else
5350 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5351 \fi
5352 \def\bbl@luapatterns#1#2{%
5353 \bbl@get@enc#1:.\@@@
5354 \setbox\z@\hbox\bgroup
5355 \begingroup
5356 \savecatcodetable\babelcatcodetablenum\relax
5357 \initcatcodetable\bbl@pattcodes\relax
5358 \catcodetable\bbl@pattcodes\relax
5359 \catcode\#=6 \catcode\$=3 \catcode\&=4 \catcode\^=7
5360 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5361 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5362 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5363 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5364 \catcode\`=12 \catcode\'=12 \catcode\"=12
5365 \input #1\relax
5366 \catcodetable\babelcatcodetablenum\relax
5367 \endgroup
5368 \def\bbl@tempa{#2}%
5369 \ifx\bbl@tempa\@empty\else
5370 \input #2\relax
5371 \fi
5372 \egroup}%
5373 \def\bbl@patterns@lua#1{%

```

```

5374 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5375   \csname l@#1\endcsname
5376   \edef\bbl@tempa{#1}%
5377 \else
5378   \csname l@#1:\f@encoding\endcsname
5379   \edef\bbl@tempa{#1:\f@encoding}%
5380 \fi\relax
5381 \namedef{lu@texhyphen@loaded@the\language}{}% Temp
5382 \ifundefined{bbl@hyphendata@the\language}%
5383   {\def\bbl@elt##1##2##3##4{%
5384     \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5385     \def\bbl@tempb{##3}%
5386     \ifx\bbl@tempb@empty\else % if not a synonymous
5387       \def\bbl@tempc{{##3}{##4}}%
5388     \fi
5389     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5390   \fi}%
5391 \bbl@languages
5392 \ifundefined{bbl@hyphendata@the\language}%
5393   {\bbl@info{No hyphenation patterns were set for\%
5394     language '\bbl@tempa'. Reported}}%
5395   {\expandafter\expandafter\expandafter\bbl@luapatterns
5396     \csname bbl@hyphendata@the\language\endcsname}}}%
5397 \endinput\fi

```

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

```

5398 \ifx\DisableBabelHook\@undefined
5399   \AddBabelHook{luatex}{everylanguage}{%
5400     \def\process@language##1##2##3{%
5401       \def\process@line####1####2 ####3 ####4 {}%
5402     \AddBabelHook{luatex}{loadpatterns}{%
5403       \input #1\relax
5404       \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5405         {{#1}}}%
5406     \AddBabelHook{luatex}{loadexceptions}{%
5407       \input #1\relax
5408       \def\bbl@tempb##1##2{{##1}{##1}}%
5409       \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5410         {\expandafter\expandafter\expandafter\bbl@tempb
5411           \csname bbl@hyphendata@the\language\endcsname}}
5412 \endinput\fi

```

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

```

5413 \begingroup
5414 \catcode`\%=12
5415 \catcode`\'=12
5416 \catcode`\%=12
5417 \catcode`\:=12
5418 \directlua{
5419   Babel.locale_props = Babel.locale_props or {}
5420   function Babel.lua_error(e, a)
5421     tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5422       e .. '}{' .. (a or '') .. '}{'}')
5423   end
5424
5425   function Babel.bytes(line)
5426     return line:gsub("(.)",
5427       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5428   end
5429
5430   function Babel.priority_in_callback(name,description)
5431     for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5432       if v == description then return i end

```

```

5433     end
5434     return false
5435 end
5436
5437 function Babel.begin_process_input()
5438     if luatexbase and luatexbase.add_to_callback then
5439         luatexbase.add_to_callback('process_input_buffer',
5440                                   Babel.bytes, 'Babel.bytes')
5441     else
5442         Babel.callback = callback.find('process_input_buffer')
5443         callback.register('process_input_buffer', Babel.bytes)
5444     end
5445 end
5446 function Babel.end_process_input ()
5447     if luatexbase and luatexbase.remove_from_callback then
5448         luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5449     else
5450         callback.register('process_input_buffer', Babel.callback)
5451     end
5452 end
5453
5454 function Babel.str_to_nodes(fn, matches, base)
5455     local n, head, last
5456     if fn == nil then return nil end
5457     for s in string.utfvalues(fn(matches)) do
5458         if base.id == 7 then
5459             base = base.replace
5460         end
5461         n = node.copy(base)
5462         n.char = s
5463         if not head then
5464             head = n
5465         else
5466             last.next = n
5467         end
5468         last = n
5469     end
5470     return head
5471 end
5472
5473 Babel.linebreaking = Babel.linebreaking or {}
5474 Babel.linebreaking.before = {}
5475 Babel.linebreaking.after = {}
5476 Babel.locale = {}
5477 function Babel.linebreaking.add_before(func, pos)
5478     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5479     if pos == nil then
5480         table.insert(Babel.linebreaking.before, func)
5481     else
5482         table.insert(Babel.linebreaking.before, pos, func)
5483     end
5484 end
5485 function Babel.linebreaking.add_after(func)
5486     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5487     table.insert(Babel.linebreaking.after, func)
5488 end
5489
5490 function Babel.addpatterns(pp, lg)
5491     local lg = lang.new(lg)
5492     local pats = lang.patterns(lg) or ''
5493     lang.clear_patterns(lg)
5494     for p in pp:gmatch('^%s+') do
5495         ss = ''

```

```

5496     for i in string.utfcharacters(p:gsub('%d', '')) do
5497         ss = ss .. '%d?' .. i
5498     end
5499     ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5500     ss = ss:gsub('%.%%d%?$', '%%.')
5501     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5502     if n == 0 then
5503         tex.sprint(
5504             [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5505             .. p .. [[]])
5506         pats = pats .. ' ' .. p
5507     else
5508         tex.sprint(
5509             [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5510             .. p .. [[]])
5511     end
5512 end
5513 lang.patterns(lg, pats)
5514 end
5515
5516 Babel.characters = Babel.characters or {}
5517 Babel.ranges = Babel.ranges or {}
5518 function Babel.hlist_has_bidi(head)
5519     local has_bidi = false
5520     local ranges = Babel.ranges
5521     for item in node.traverse(head) do
5522         if item.id == node.id'glyph' then
5523             local itemchar = item.char
5524             local chardata = Babel.characters[itemchar]
5525             local dir = chardata and chardata.d or nil
5526             if not dir then
5527                 for nn, et in ipairs(ranges) do
5528                     if itemchar < et[1] then
5529                         break
5530                     elseif itemchar <= et[2] then
5531                         dir = et[3]
5532                         break
5533                     end
5534                 end
5535             end
5536             if dir and (dir == 'al' or dir == 'r') then
5537                 has_bidi = true
5538             end
5539         end
5540     end
5541     return has_bidi
5542 end
5543 function Babel.set_chranges_b (script, chrng)
5544     if chrng == '' then return end
5545     texio.write('Replacing ' .. script .. ' script ranges')
5546     Babel.script_blocks[script] = {}
5547     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5548         table.insert(
5549             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5550     end
5551 end
5552
5553 function Babel.discard_sublr(str)
5554     if str:find( [[\string\indexentry]] ) and
5555        str:find( [[\string\babelsublr]] ) then
5556         str = str:gsub( [[\string\babelsublr%s*(%b{})]] ,
5557             function(m) return m:sub(2,-2) end )
5558     end

```

```

5559     return str
5560 end
5561 }
5562 \endgroup
5563 \ifx\newattribute\undefined\else % Test for plain
5564   \newattribute\bbbl@attr@locale % DL4
5565   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbbl@attr@locale' }
5566   \AddBabelHook{luatex}{beforeextras}{%
5567     \setattribute\bbbl@attr@locale\localeid}
5568 \fi
5569 %
5570 \def\BabelStringsDefault{unicode}
5571 \let\luabbbl@stop\relax
5572 \AddBabelHook{luatex}{encodedcommands}{%
5573   \def\bbbl@tempa{utf8}\def\bbbl@tempb{#1}%
5574   \ifx\bbbl@tempa\bbbl@tempb\else
5575     \directlua{Babel.begin_process_input()}%
5576     \def\luabbbl@stop{%
5577       \directlua{Babel.end_process_input()}}%
5578   \fi}%
5579 \AddBabelHook{luatex}{stopcommands}{%
5580   \luabbbl@stop
5581   \let\luabbbl@stop\relax}
5582 %
5583 \AddBabelHook{luatex}{patterns}{%
5584   \@ifundefined{bbbl@hyphendata@the\language}%
5585     {\def\bbbl@elt##1##2##3##4{%
5586       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5587       \def\bbbl@tempb{##3}%
5588       \ifx\bbbl@tempb\@empty\else % if not a synonymous
5589         \def\bbbl@tempc{{##3}{##4}}%
5590         \fi
5591         \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%
5592       \fi}%
5593   \bbbl@languages
5594   \@ifundefined{bbbl@hyphendata@the\language}%
5595     {\bbbl@info{No hyphenation patterns were set for\%
5596       language '#2'. Reported}}%
5597     {\expandafter\expandafter\expandafter\bbbl@luapatterns
5598       \csname bbl@hyphendata@the\language\endcsname}}}%
5599   \@ifundefined{bbbl@patterns@}{}%
5600   \begingroup
5601     \bbbl@xin@{,\number\language,}{,\bbbl@pttnlist}%
5602     \ifin@else
5603       \ifx\bbbl@patterns@\@empty\else
5604         \directlua{ Babel.addpatterns(
5605           [[\bbbl@patterns@]], \number\language) }%
5606       \fi
5607       \@ifundefined{bbbl@patterns@#1}%
5608       \@empty
5609       {\directlua{ Babel.addpatterns(
5610         [[\space\csname bbl@patterns@#1\endcsname]],
5611         \number\language) }}%
5612       \xdef\bbbl@pttnlist{\bbbl@pttnlist\number\language,}%
5613     \fi
5614   \endgroup}%
5615   \bbbl@exp{%
5616     \bbbl@ifunset{bbbl@prehc@languagename}}}%
5617     {\bbbl@ifblank{\bbbl@cs{prehc@languagename}}}%
5618     {\prehyphenchar=\bbbl@cl{prehc}\relax}}}}

```

\babelpatterns This macro adds patterns. Two macros are used to store them: \bbbl@patterns@ for the global ones and \bbbl@patterns@*language* for language ones. We make sure there is a space

between words when multiple commands are used.

```

5619 \@onlypreamble\babelpatterns
5620 \AtEndOfPackage{%
5621   \newcommand\babelpatterns[2][\@empty]{%
5622     \ifx\bbbl@patterns@relax
5623       \let\bbbl@patterns@\@empty
5624     \fi
5625     \ifx\bbbl@pttnlist\@empty\else
5626       \bbbl@warning{%
5627         You must not intermingle \string\selectlanguage\space and\\%
5628         \string\babelpatterns\space or some patterns will not\\%
5629         be taken into account. Reported}%
5630     \fi
5631     \ifx\@empty#1%
5632       \protected@edef\bbbl@patterns@\bbbl@patterns@\space#2}%
5633     \else
5634       \edef\bbbl@tempb{\zap@space#1 \@empty}%
5635       \bbbl@for\bbbl@tempa\bbbl@tempb{%
5636         \bbbl@fixname\bbbl@tempa
5637         \bbbl@iflanguage\bbbl@tempa{%
5638           \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
5639             \@ifundefined{bbbl@patterns@\bbbl@tempa}%
5640               \@empty
5641               {\csname bbl@patterns@\bbbl@tempa\endcsname\space}%
5642             #2}}}%
5643     \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.

```

5644 \def\bbbl@intraspace#1 #2 #3\@{
5645   \directlua{
5646     Babel.intraspaces = Babel.intraspaces or {}
5647     Babel.intraspaces['\csname bbl@sbcpr@languagename\endcsname'] = %
5648       {b = #1, p = #2, m = #3}
5649     Babel.locale_props[\the\localeid].intraspace = %
5650       {b = #1, p = #2, m = #3}
5651   }}
5652 \def\bbbl@intrapenalty#1\@{
5653   \directlua{
5654     Babel.intrapenalties = Babel.intrapenalties or {}
5655     Babel.intrapenalties['\csname bbl@sbcpr@languagename\endcsname'] = #1
5656     Babel.locale_props[\the\localeid].intrapenalty = #1
5657   }}
5658 \begingroup
5659 \catcode`\%=12
5660 \catcode`\&=14
5661 \catcode`\'=12
5662 \catcode`\-=12
5663 \gdef\bbbl@seaintraspace{&
5664   \let\bbbl@seaintraspace\relax
5665   \directlua{
5666     Babel.sea_enabled = true
5667     Babel.sea_ranges = Babel.sea_ranges or {}
5668     function Babel.set_chranges (script, chrng)
5669       local c = 0
5670       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5671         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5672         c = c + 1

```

```

5673     end
5674 end
5675 function Babel.sea_disc_to_space (head)
5676     local sea_ranges = Babel.sea_ranges
5677     local last_char = nil
5678     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5679     for item in node.traverse(head) do
5680         local i = item.id
5681         if i == node.id'glyph' then
5682             last_char = item
5683         elseif i == 7 and item.subtype == 3 and last_char
5684             and last_char.char > 0x0C99 then
5685             quad = font.getfont(last_char.font).size
5686             for lg, rg in pairs(sea_ranges) do
5687                 if last_char.char > rg[1] and last_char.char < rg[2] then
5688                     lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5689                     local intraspace = Babel.intraspaces[lg]
5690                     local intrapenalty = Babel.intrapenalties[lg]
5691                     local n
5692                     if intrapenalty ~= 0 then
5693                         n = node.new(14, 0)      &% penalty
5694                         n.penalty = intrapenalty
5695                         node.insert_before(head, item, n)
5696                     end
5697                     n = node.new(12, 13)      &% (glue, spaceskip)
5698                     node.setglue(n, intraspace.b * quad,
5699                                 intraspace.p * quad,
5700                                 intraspace.m * quad)
5701                     node.insert_before(head, item, n)
5702                     node.remove(head, item)
5703                 end
5704             end
5705         end
5706     end
5707 end
5708 }&
5709 \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.

```

5710 \catcode`\%=14
5711 \gdef\bbl@cjk intraspace{%
5712     \let\bbl@cjk intraspace\relax
5713     \directlua{
5714         require('babel-data-cjk.lua')
5715         Babel.cjk_enabled = true
5716         function Babel.cjk_linebreak(head)
5717             local GLYPH = node.id'glyph'
5718             local last_char = nil
5719             local quad = 655360      % 10 pt = 655360 = 10 * 65536
5720             local last_class = nil
5721             local last_lang = nil
5722             for item in node.traverse(head) do
5723                 if item.id == GLYPH then
5724                     local lang = item.lang
5725                     local LOCALE = node.get_attribute(item,
5726                         Babel.attr_locale)

```

```

5727     local props = Babel.locale_props[LOCALE] or {}
5728     local class = Babel.cjk_class[item.char].c
5729     if props.cjk_quotes and props.cjk_quotes[item.char] then
5730         class = props.cjk_quotes[item.char]
5731     end
5732     if class == 'cp' then class = 'cl' % )] as CL
5733     elseif class == 'id' then class = 'I'
5734     elseif class == 'cj' then class = 'I' % loose
5735     end
5736     local br = 0
5737     if class and last_class and Babel.cjk_breaks[last_class][class] then
5738         br = Babel.cjk_breaks[last_class][class]
5739     end
5740     if br == 1 and props.linebreak == 'c' and
5741         lang ~= \the\l@nohyphenation\space and
5742         last_lang ~= \the\l@nohyphenation then
5743         local intrapenalty = props.intrapenalty
5744         if intrapenalty ~= 0 then
5745             local n = node.new(14, 0)      % penalty
5746             n.penalty = intrapenalty
5747             node.insert_before(head, item, n)
5748         end
5749         local intraspace = props.intraspace
5750         local n = node.new(12, 13)      % (glue, spaceskip)
5751         node.setglue(n, intraspace.b * quad,
5752             intraspace.p * quad,
5753             intraspace.m * quad)
5754         node.insert_before(head, item, n)
5755     end
5756     if font.getfont(item.font) then
5757         quad = font.getfont(item.font).size
5758     end
5759     last_class = class
5760     last_lang = lang
5761     else % if penalty, glue or anything else
5762         last_class = nil
5763     end
5764     end
5765     lang.hyphenate(head)
5766 end
5767 }%
5768 \bbl@luahyphenate}
5769 \gdef\bbl@luahyphenate{%
5770 \let\bbl@luahyphenate\relax
5771 \directlua{
5772     luatexbase.add_to_callback('hyphenate',
5773     function (head, tail)
5774         if Babel.linebreaking.before then
5775             for k, func in ipairs(Babel.linebreaking.before) do
5776                 func(head)
5777             end
5778         end
5779         lang.hyphenate(head)
5780         if Babel.cjk_enabled then
5781             Babel.cjk_linebreak(head)
5782         end
5783         if Babel.linebreaking.after then
5784             for k, func in ipairs(Babel.linebreaking.after) do
5785                 func(head)
5786             end
5787         end
5788         if Babel.set_hboxed then
5789             Babel.set_hboxed(head)

```

```

5790     end
5791     if Babel.sea_enabled then
5792         Babel.sea_disc_to_space(head)
5793     end
5794 end,
5795 'Babel.hyphenate')
5796 }}
5797 \endgroup
5798 %
5799 \def\bbl@provide@intraspace{%
5800   \bbl@ifunset{\bbl@intsp@{language}}{%
5801     {\expandafter\ifx\csname\bbl@intsp@{language}\endcsname\@empty\else
5802       \bbl@xin@{c}{\bbl@cl{lnbrk}}}%
5803     \ifin@           % cjk
5804       \bbl@cjk@intraspace
5805       \directlua{
5806         Babel.locale_props = Babel.locale_props or {}
5807         Babel.locale_props[\the\localeid].linebreak = 'c'
5808       }%
5809       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5810       \ifx\bbl@KVP@intrapenalty\@nnil
5811         \bbl@intrapenalty0\@{}
5812       \fi
5813     \else           % sea
5814       \bbl@sea@intraspace
5815       \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5816       \directlua{
5817         Babel.sea_ranges = Babel.sea_ranges or {}
5818         Babel.set_chranges('\bbl@cl{sbc}{',
5819           '\bbl@cl{chrng}')
5820       }%
5821       \ifx\bbl@KVP@intrapenalty\@nnil
5822         \bbl@intrapenalty0\@{}
5823       \fi
5824     \fi
5825   \fi
5826   \ifx\bbl@KVP@intrapenalty\@nnil\else
5827     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@{}
5828   \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`.

```

5829 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5830 \def\bblar@chars{%
5831   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5832   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5833   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5834 \def\bblar@elongated{%
5835   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5836   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5837   0649,064A}
5838 \begingroup
5839   \catcode`\_ =11 \catcode`\:=11
5840   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5841 \endgroup
5842 \gdef\bbl@arabicjust{%
5843   \let\bbl@arabicjust\relax
5844   \newattribute\bblar@kashida
5845   \directlua{ Babel.attr_kashida = luatexbase.registernumber' \bblar@kashida' }%
5846   \bblar@kashida=\z@
5847   \bbl@patchfont{\bbl@parsejalt}}%

```

```

5848 \directlua{
5849   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5850   Babel.arabic.elong_map[\the\localeid] = {}
5851   luatexbase.add_to_callback('post_linebreak_filter',
5852     Babel.arabic.justify, 'Babel.arabic.justify')
5853   luatexbase.add_to_callback('hpack_filter',
5854     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5855 }%

  Save both node lists to make replacement.

5856 \def\bblar@fetchjalt#1#2#3#4{%
5857   \bbl@exp{\bbl@foreach{#1}}{%
5858     \bbl@ifunset{bblar@JE@##1}%
5859     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5860     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5861   \directlua{%
5862     local last = nil
5863     for item in node.traverse(tex.box[0].head) do
5864       if item.id == node.id'glyph' and item.char > 0x600 and
5865         not (item.char == 0x200D) then
5866         last = item
5867       end
5868     end
5869     Babel.arabic.#3['##1#4'] = last.char
5870   }}

```

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

```

5871 \gdef\bbl@parsejalt{%
5872   \ifx\addfontfeature\undefined\else
5873     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5874     \ifin@
5875       \directlua{%
5876         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5877           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5878           tex.print([[string\cswf\space bbl@parsejalti\endcswf]])
5879         end
5880       }%
5881     \fi
5882   \fi}
5883 \gdef\bbl@parsejalti{%
5884   \begingroup
5885     \let\bbl@parsejalt\relax % To avoid infinite loop
5886     \edef\bbl@tempb{\fontid\font}%
5887     \bblar@nofswarn
5888     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5889     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5890     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5891     \addfontfeature{RawFeature+=jalt}%
5892     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5893     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5894     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5895     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5896     \directlua{%
5897       for k, v in pairs(Babel.arabic.from) do
5898         if Babel.arabic.dest[k] and
5899           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5900           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5901             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5902         end
5903       end
5904     }%
5905   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```
5906 \begingroup
5907 \catcode`#=11
5908 \catcode`~=11
5909 \directlua{
5910
5911 Babel.arabic = Babel.arabic or {}
5912 Babel.arabic.from = {}
5913 Babel.arabic.dest = {}
5914 Babel.arabic.justify_factor = 0.95
5915 Babel.arabic.justify_enabled = true
5916 Babel.arabic.kashida_limit = -1
5917
5918 function Babel.arabic.justify(head)
5919   if not Babel.arabic.justify_enabled then return head end
5920   for line in node.traverse_id(node.id'hlist', head) do
5921     Babel.arabic.justify_hlist(head, line)
5922   end
5923   % In case the very first item is a line (eg, in \vbox):
5924   while head.prev do head = head.prev end
5925   return head
5926 end
5927
5928 function Babel.arabic.justify_hbox(head, gc, size, pack)
5929   local has_inf = false
5930   if Babel.arabic.justify_enabled and pack == 'exactly' then
5931     for n in node.traverse_id(12, head) do
5932       if n.stretch_order > 0 then has_inf = true end
5933     end
5934     if not has_inf then
5935       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5936     end
5937   end
5938   return head
5939 end
5940
5941 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5942   local d, new
5943   local k_list, k_item, pos_inline
5944   local width, width_new, full, k_curr, wt_pos, goal, shift
5945   local subst_done = false
5946   local elong_map = Babel.arabic.elong_map
5947   local cnt
5948   local last_line
5949   local GLYPH = node.id'glyph'
5950   local KASHIDA = Babel.attr_kashida
5951   local LOCALE = Babel.attr_locale
5952
5953   if line == nil then
5954     line = {}
5955     line.glue_sign = 1
5956     line.glue_order = 0
5957     line.head = head
5958     line.shift = 0
5959     line.width = size
5960   end
5961
5962   % Exclude last line. todo. But-- it discards one-word lines, too!
5963   % ? Look for glue = 12:15
5964   if (line.glue_sign == 1 and line.glue_order == 0) then
5965     elongs = {} % Stores elongated candidates of each line
5966     k_list = {} % And all letters with kashida
5967     pos_inline = 0 % Not yet used
```

```

5968
5969 for n in node.traverse_id(GLYPH, line.head) do
5970     pos_inline = pos_inline + 1 % To find where it is. Not used.
5971
5972     % Elongated glyphs
5973     if elong_map then
5974         local locale = node.get_attribute(n, LOCALE)
5975         if elong_map[locale] and elong_map[locale][n.font] and
5976             elong_map[locale][n.font][n.char] then
5977             table.insert(elongs, {node = n, locale = locale} )
5978             node.set_attribute(n.prev, KASHIDA, 0)
5979         end
5980     end
5981
5982     % Tatwil. First create a list of nodes marked with kashida. The
5983     % rest of nodes can be ignored. The list of used weights is build
5984     % when transforms with the key kashida= are declared.
5985     if Babel.kashida_wts then
5986         local k_wt = node.get_attribute(n, KASHIDA)
5987         if k_wt > 0 then % todo. parameter for multi inserts
5988             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5989         end
5990     end
5991
5992 end % of node.traverse_id
5993
5994 if #elongs == 0 and #k_list == 0 then goto next_line end
5995 full = line.width
5996 shift = line.shift
5997 goal = full * Babel.arabic.justify_factor % A bit crude
5998 width = node.dimensions(line.head) % The 'natural' width
5999
6000 % == Elongated ==
6001 % Original idea taken from 'chickenize'
6002 while (#elongs > 0 and width < goal) do
6003     subst_done = true
6004     local x = #elongs
6005     local curr = elongs[x].node
6006     local oldchar = curr.char
6007     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6008     width = node.dimensions(line.head) % Check if the line is too wide
6009     % Substitute back if the line would be too wide and break:
6010     if width > goal then
6011         curr.char = oldchar
6012         break
6013     end
6014     % If continue, pop the just substituted node from the list:
6015     table.remove(elongs, x)
6016 end
6017
6018 % == Tatwil ==
6019 % Traverse the kashida node list so many times as required, until
6020 % the line is filled. The first pass adds a tatweel after each
6021 % node with kashida in the line, the second pass adds another one,
6022 % and so on. In each pass, add first the kashida with the highest
6023 % weight, then with lower weight and so on.
6024 if #k_list == 0 then goto next_line end
6025
6026 width = node.dimensions(line.head) % The 'natural' width
6027 k_curr = #k_list % Traverse backwards, from the end
6028 wt_pos = 1
6029
6030 while width < goal do

```

```

6031     subst_done = true
6032     k_item = k_list[k_curr].node
6033     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6034         d = node.copy(k_item)
6035         d.char = 0x0640
6036         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6037         d.xoffset = 0
6038         line.head, new = node.insert_after(line.head, k_item, d)
6039         width_new = node.dimensions(line.head)
6040         if width > goal or width == width_new then
6041             node.remove(line.head, new) % Better compute before
6042             break
6043         end
6044         if Babel.fix_diacr then
6045             Babel.fix_diacr(k_item.next)
6046         end
6047         width = width_new
6048     end
6049     if k_curr == 1 then
6050         k_curr = #k_list
6051         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6052     else
6053         k_curr = k_curr - 1
6054     end
6055 end
6056
6057 % Limit the number of tatweel by removing them. Not very efficient,
6058 % but it does the job in a quite predictable way.
6059 if Babel.arabic.kashida_limit > -1 then
6060     cnt = 0
6061     for n in node.traverse_id(GLYPH, line.head) do
6062         if n.char == 0x0640 then
6063             cnt = cnt + 1
6064             if cnt > Babel.arabic.kashida_limit then
6065                 node.remove(line.head, n)
6066             end
6067         else
6068             cnt = 0
6069         end
6070     end
6071 end
6072
6073 ::next_line::
6074
6075 % Must take into account marks and ins, see luatex manual.
6076 % Have to be executed only if there are changes. Investigate
6077 % what's going on exactly.
6078 if subst_done and not gc then
6079     d = node.hpack(line.head, full, 'exactly')
6080     d.shift = shift
6081     node.insert_before(head, line, d)
6082     node.remove(head, line)
6083 end
6084 end % if process line
6085 end
6086 }
6087 \endgroup
6088 \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.

```

6089 \def\bbl@scr@node@list{%
6090   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6091   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6092 \ifnum\bbl@bidimode=102 % bidi-r
6093   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6094 \fi
6095 \def\bbl@set@renderer{%
6096   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6097   \ifin@
6098     \let\bbl@unset@renderer\relax
6099   \else
6100     \bbl@exp{%
6101       \def\\bbl@unset@renderer{%
6102         \def<g__fontspec_default_fontopts_clist>{%
6103           [g__fontspec_default_fontopts_clist]}%
6104         \def<g__fontspec_default_fontopts_clist>{%
6105           Renderer=Harfbuzz,[g__fontspec_default_fontopts_clist]}%
6106       \fi}
6107 <@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.

```

6108 \directlua{% DL6
6109 Babel.script_blocks = {
6110   ['dflt'] = {},
6111   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6112             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6113   ['Armn'] = {{0x0530, 0x058F}},
6114   ['Beng'] = {{0x0980, 0x09FF}},
6115   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6116   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6117   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6118             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6119   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6120   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6121             {0xAB00, 0xAB2F}},
6122   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6123   % Don't follow strictly Unicode, which places some Coptic letters in
6124   % the 'Greek and Coptic' block
6125   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6126   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6127             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6128             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6129             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6130             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6131             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6132   ['Hebr'] = {{0x0590, 0x05FF},
6133             {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6134   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6135             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6136   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},

```

```

6137 ['Knda'] = {{0x0C80, 0x0CFF}},
6138 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6139             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6140             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6141 ['Lao'] = {{0x0E80, 0x0EFF}},
6142 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6143             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6144             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6145 ['Mahj'] = {{0x11150, 0x1117F}},
6146 ['Mlym'] = {{0x0D00, 0x0D7F}},
6147 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6148 ['Orya'] = {{0x0B00, 0x0B7F}},
6149 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6150 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6151 ['Taml'] = {{0x0B80, 0x0BFF}},
6152 ['Telu'] = {{0x0C00, 0x0C7F}},
6153 ['Tfng'] = {{0x2D30, 0x2D7F}},
6154 ['Thai'] = {{0x0E00, 0x0E7F}},
6155 ['Tibt'] = {{0x0F00, 0x0FFF}},
6156 ['Vaii'] = {{0xA500, 0xA63F}},
6157 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6158 }
6159
6160 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6161 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6162 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6163
6164 function Babel.locale_map(head)
6165   if not Babel.locale_mapped then return head end
6166
6167   local LOCALE = Babel.attr_locale
6168   local GLYPH = node.id('glyph')
6169   local inmath = false
6170   local toloc_save
6171   for item in node.traverse(head) do
6172     local toloc
6173     if not inmath and item.id == GLYPH then
6174       % Optimization: build a table with the chars found
6175       if Babel.chr_to_loc[item.char] then
6176         toloc = Babel.chr_to_loc[item.char]
6177       else
6178         for lc, maps in pairs(Babel.loc_to_scr) do
6179           for _, rg in pairs(maps) do
6180             if item.char >= rg[1] and item.char <= rg[2] then
6181               Babel.chr_to_loc[item.char] = lc
6182               toloc = lc
6183               break
6184             end
6185           end
6186         end
6187         % Treat composite chars in a different fashion, because they
6188         % 'inherit' the previous locale.
6189         if (item.char >= 0x0300 and item.char <= 0x036F) or
6190            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6191            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6192           Babel.chr_to_loc[item.char] = -2000
6193           toloc = -2000
6194         end
6195         if not toloc then
6196           Babel.chr_to_loc[item.char] = -1000
6197         end
6198       end
6199       if toloc == -2000 then

```

```

6200         toloc = toloc_save
6201     elseif toloc == -1000 then
6202         toloc = nil
6203     end
6204     if toloc and Babel.locale_props[toloc] and
6205         Babel.locale_props[toloc].letters and
6206         tex.getcatcode(item.char) \string~= 11 then
6207         toloc = nil
6208     end
6209     if toloc and Babel.locale_props[toloc].script
6210         and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6211         and Babel.locale_props[toloc].script ==
6212         Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6213         toloc = nil
6214     end
6215     if toloc then
6216         if Babel.locale_props[toloc].lg then
6217             item.lang = Babel.locale_props[toloc].lg
6218             node.set_attribute(item, LOCALE, toloc)
6219         end
6220         if Babel.locale_props[toloc]['/'..item.font] then
6221             item.font = Babel.locale_props[toloc]['/'..item.font]
6222         end
6223     end
6224     toloc_save = toloc
6225     elseif not inmath and item.id == 7 then % Apply recursively
6226         item.replace = item.replace and Babel.locale_map(item.replace)
6227         item.pre      = item.pre and Babel.locale_map(item.pre)
6228         item.post      = item.post and Babel.locale_map(item.post)
6229     elseif item.id == node.id'math' then
6230         inmath = (item.subtype == 0)
6231     end
6232 end
6233 return head
6234 end
6235 }

```

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

```

6236 \newcommand\babelcharproperty[1]{%
6237   \count@=#1\relax
6238   \ifvmode
6239     \expandafter\bbl@chprop
6240   \else
6241     \bbl@error{charproperty-only-vertical}{}{}{}%
6242   \fi}
6243 \newcommand\bbl@chprop[3][\the\count@]{%
6244   \@tempcnta=#1\relax
6245   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6246   {\bbl@error{unknown-char-property}{}{#2}{}%
6247   {}%
6248   \loop
6249     \bbl@cs{chprop@#2}{#3}%
6250   \ifnum\count@<\@tempcnta
6251     \advance\count@\@ne
6252   \repeat}
6253 %
6254 \def\bbl@chprop@direction#1{%
6255   \directlua{
6256     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6257     Babel.characters[\the\count@]['d'] = '#1'
6258   }}
6259 \let\bbl@chprop@bc\bbl@chprop@direction

```

```

6260 %
6261 \def\bbl@chprop@mirror#1{%
6262   \directlua{
6263     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6264     Babel.characters[\the\count@]['m'] = '\number#1'
6265   }}
6266 \let\bbl@chprop@bmg\bbl@chprop@mirror
6267 %
6268 \def\bbl@chprop@linebreak#1{%
6269   \directlua{
6270     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6271     Babel.cjk_characters[\the\count@]['c'] = '#1'
6272   }}
6273 \let\bbl@chprop@lb\bbl@chprop@linebreak
6274 %
6275 \def\bbl@chprop@locale#1{%
6276   \directlua{
6277     Babel.chr_to_loc = Babel.chr_to_loc or {}
6278     Babel.chr_to_loc[\the\count@] =
6279       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6280   }}

```

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.

```

6281 \directlua{% DL7
6282   Babel.nohyphenation = \the\l@nohyphenation
6283 }

```

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, $\text{pre}=\{1\}\{1\}$ becomes $\text{function}(m)$ `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 $\text{function}(m)$ `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).

```

6284 \begingroup
6285 \catcode`\~ = 12
6286 \catcode`\% = 12
6287 \catcode`\& = 14
6288 \catcode`\| = 12
6289 \gdef\babelprehyphenation{%&
6290   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}[]]}
6291 \gdef\babelposthyphenation{%&
6292   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}[]]}
6293 %
6294 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6295   \ifcase#1
6296     \bbl@activateprehyphen
6297   \or
6298     \bbl@activateposthyphen
6299   \fi
6300   \begingroup
6301     \def\babeltempa{\bbl@add@list\babeltempb}%&
6302     \let\babeltempb\empty
6303     \def\bbl@tempa{#5}%&
6304     \bbl@replace\bbl@tempa{,}{ ,}%& TODO. Ugly trick to preserve {}
6305     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6306       \bbl@ifsamestring{##1}{remove}%&
6307       {\bbl@add@list\babeltempb{nil}}}%&
6308     {\directlua{
6309       local rep = [=##1]=]
6310       local three_args = '%S'..%S*([%-d%.%a{}|]+)%S+([%-d%.%a{}|]+)%S+([%-d%.%a{}|]+)'

```

```

6311      &% Numeric passes directly: kern, penalty...
6312      rep = rep:gsup('^%s*(remove)%s*$', 'remove = true')
6313      rep = rep:gsup('^%s*(insert)%s*', 'insert = true, ')
6314      rep = rep:gsup('^%s*(after)%s*', 'after = true, ')
6315      rep = rep:gsup('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6316      rep = rep:gsup('node%s*=%s*(%a+)%s*([^\s,]*)', Babel.capture_node)
6317      rep = rep:gsup(' (norule)' .. three_args,
6318        'norule = {' .. '%2, %3, %4' .. '}')
6319      if #1 == 0 or #1 == 2 then
6320        rep = rep:gsup(' (space)' .. three_args,
6321          'space = {' .. '%2, %3, %4' .. '}')
6322        rep = rep:gsup(' (spacefactor)' .. three_args,
6323          'spacefactor = {' .. '%2, %3, %4' .. '}')
6324        rep = rep:gsup('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6325      &% Transform values
6326      rep, n = rep:gsup(' ({[%a%-%.]+}|([%a%_%.]+))',
6327        function(v,d)
6328          return string.format (
6329            '{\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6330            v,
6331            load( 'return Babel.locale_props'..
6332              '[\the\csname bbl@id@@#3\endcsname].' .. d)() )
6333          end )
6334      rep, n = rep:gsup(' ({[%a%-%.]+}|([%-d%.]+))',
6335        '{\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6336      end
6337      if #1 == 1 then
6338        rep = rep:gsup(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6339        rep = rep:gsup(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6340        rep = rep:gsup(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6341      end
6342      tex.print([\string\babeltempa{[]] .. rep .. [[]]])
6343    }&%
6344    \bbl@foreach\babeltempb{&%
6345      \bbl@forkv{##1}{&%
6346        \in@{,###1,},{,nil,step,data,remove,insert,string,no,pre,no,&%
6347          post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6348        \ifin@else
6349          \bbl@error{bad-transform-option}{###1}{}&%
6350        \fi}&%
6351      \let\bbl@kv@attribute\relax
6352      \let\bbl@kv@label\relax
6353      \let\bbl@kv@fonts\@empty
6354      \let\bbl@kv@prepend\relax
6355      \bbl@forkv{#2}{\bbl@csarg\edef{kv{##1}{##2}}&%
6356      \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6357      \ifx\bbl@kv@attribute\relax
6358        \ifx\bbl@kv@label\relax\else
6359          \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6360          \bbl@replace\bbl@kv@fonts{ }{,}&%
6361          \edef\bbl@kv@attribute{\bbl@ATR@{\bbl@kv@label @#3@\bbl@kv@fonts}&%
6362            \count@ \z@
6363            \def\bbl@elt##1##2##3{&%
6364              \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6365              {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6366                {\count@ \@ne}&%
6367                {\bbl@error{font-conflict-transforms}{}}}&%
6368              }&%
6369            \bbl@transfont@list
6370            \ifnum\count@=\z@
6371              \bbl@exp{\global\bbl@add\bbl@transfont@list
6372                {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6373            \fi

```

```

6374      \bbl@ifunset{\bbl@kv@attribute}&%
6375      {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6376      }&%
6377      \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6378      \fi
6379  \else
6380      \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6381      \fi
6382  \directlua{
6383      local lbkr = Babel.linebreaking.replacements[#1]
6384      local u = unicode.utf8
6385      local id, attr, label
6386      if #1 == 0 then
6387          id = \the\csname bbl@id@#3\endcsname\space
6388      else
6389          id = \the\csname l@#3\endcsname\space
6390      end
6391      \ifx\bbl@kv@attribute\relax
6392          attr = -1
6393      \else
6394          attr = luatexbase.registernumber'\bbl@kv@attribute'
6395      \fi
6396      \ifx\bbl@kv@label\relax\else &% Same refs:
6397          label = [==[\bbl@kv@label]==]
6398      \fi
6399      &% Convert pattern:
6400      local patt = string.gsub([==[#4]==], '%s', '')
6401      if #1 == 0 then
6402          patt = string.gsub(patt, '|', ' ')
6403      end
6404      if not u.find(patt, '()', nil, true) then
6405          patt = '()' .. patt .. '()'
6406      end
6407      if #1 == 1 then
6408          patt = string.gsub(patt, '%(%)^', '^()')
6409          patt = string.gsub(patt, '%$(%)', '()$')
6410      end
6411      patt = u.gsub(patt, '{(.)}',
6412          function (n)
6413              return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6414          end)
6415      patt = u.gsub(patt, '{(%x%x%x%x+)}',
6416          function (n)
6417              return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6418          end)
6419      lbkr[id] = lbkr[id] or {}
6420      table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6421          { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6422      }&%
6423  \endgroup}
6424 \endgroup
6425 %
6426 \let\bbl@transfont@list\@empty
6427 \def\bbl@settransfont{%
6428     \global\let\bbl@settransfont\relax % Execute only once
6429     \gdef\bbl@transfont{%
6430         \def\bbl@elt####1####2####3{%
6431             \bbl@ifblank{####3}%
6432             {\count@tw@}% Do nothing if no fonts
6433             {\count@z@
6434                 \bbl@vforeach{####3}{%
6435                     \def\bbl@tempd{#####1}%
6436                     \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%

```

```

6437         \ifx\bbl@tempd\bbl@tempe
6438         \count@\@ne
6439         \else\ifx\bbl@tempd\bbl@transfam
6440         \count@\@ne
6441         \fi\fi}%
6442     \ifcase\count@
6443     \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
6444     \or
6445     \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
6446     \fi}}%
6447     \bbl@transfont@list}%
6448 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6449 \gdef\bbl@transfam{-unknown-}%
6450 \bbl@foreach\bbl@font@fams{%
6451     \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6452     \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6453     {\xdef\bbl@transfam{##1}}%
6454     {}}}%
6455 %
6456 \DeclareRobustCommand\enablelocaletransform[1]{%
6457     \bbl@ifunset{bbl@ATR@#1@\language @}%
6458     {\bbl@error{transform-not-available}{#1}{}}}%
6459     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6460 \DeclareRobustCommand\disablelocaletransform[1]{%
6461     \bbl@ifunset{bbl@ATR@#1@\language @}%
6462     {\bbl@error{transform-not-available-b}{#1}{}}}%
6463     {\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`.

```

6464 \def\bbl@activateposthyphen{%
6465     \let\bbl@activateposthyphen\relax
6466     \ifx\bbl@attr@hboxed\undefined
6467         \newattribute\bbl@attr@hboxed
6468     \fi
6469     \directlua{
6470         require('babel-transforms.lua')
6471         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6472     }}
6473 \def\bbl@activateprehyphen{%
6474     \let\bbl@activateprehyphen\relax
6475     \ifx\bbl@attr@hboxed\undefined
6476         \newattribute\bbl@attr@hboxed
6477     \fi
6478     \directlua{
6479         require('babel-transforms.lua')
6480         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6481     }}
6482 \newcommand\SetTransformValue[3]{%
6483     \directlua{
6484         Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6485     }}

```

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.

```

6486 \newcommand\ShowBabelTransforms[1]{%
6487     \bbl@activateprehyphen
6488     \bbl@activateposthyphen
6489     \begingroup
6490     \directlua{ Babel.show_transforms = true }%
6491     \setbox\z@\vbox{#1}%
6492     \directlua{ Babel.show_transforms = false }%
6493     \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.

```
6494 \newcommand\localeprehyphenation[1]{%
6495 \directlua{ Babel.string_prehyphenation([==[#1]==], \the\localeid) }}
```

10.11.Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before luaotfload is applied, which is loaded by default by \TeX . Just in case, consider the possibility it has not been loaded.

```
6496 \def\bbl@activate@preotf{%
6497 \let\bbl@activate@preotf\relax % only once
6498 \directlua{
6499   function Babel.pre_otfload_v(head)
6500     if Babel.numbers and Babel.digits_mapped then
6501       head = Babel.numbers(head)
6502     end
6503     if Babel.bidi_enabled then
6504       head = Babel.bidi(head, false, dir)
6505     end
6506     return head
6507   end
6508   %
6509   function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6510     if Babel.numbers and Babel.digits_mapped then
6511       head = Babel.numbers(head)
6512     end
6513     if Babel.bidi_enabled then
6514       head = Babel.bidi(head, false, dir)
6515     end
6516     return head
6517   end
6518   %
6519   luatexbase.add_to_callback('pre_linebreak_filter',
6520     Babel.pre_otfload_v,
6521     'Babel.pre_otfload_v',
6522     Babel.priority_in_callback('pre_linebreak_filter',
6523       'luaotfload.node_processor') or nil)
6524   %
6525   luatexbase.add_to_callback('hpack_filter',
6526     Babel.pre_otfload_h,
6527     'Babel.pre_otfload_h',
6528     Babel.priority_in_callback('hpack_filter',
6529       'luaotfload.node_processor') or nil)
6530 }
```

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.

```
6531 \breakafterdirmode=1
6532 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6533 \let\bbl@beforeforeign\leavevmode
6534 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6535 \RequirePackage{luatexbase}
6536 \bbl@activate@preotf
6537 \directlua{
6538   require('babel-data-bidi.lua')
6539   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
```



```

6540     require('babel-bidi-basic.lua')
6541     \or
6542     require('babel-bidi-basic-r.lua')
6543     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6544     table.insert(Babel.ranges, {0xF0000, 0xFFFFFD, 'on'})
6545     table.insert(Babel.ranges, {0x100000, 0x10FFFFD, 'on'})
6546     \fi}
6547 \newattribute\bbl@attr@dir
6548 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6549 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6550 \fi
6551 %
6552 \chardef\bbl@thetextdir\z@
6553 \chardef\bbl@thepardir\z@
6554 \def\bbl@getluadir#1{%
6555   \directlua{
6556     if tex.#ldir == 'TLT' then
6557       tex.sprint('0')
6558     elseif tex.#ldir == 'TRT' then
6559       tex.sprint('1')
6560     else
6561       tex.sprint('0')
6562     end}}
6563 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6564   \ifcase#3\relax
6565     \ifcase\bbl@getluadir{#1}\relax\else
6566       #2 TLT\relax
6567     \fi
6568   \else
6569     \ifcase\bbl@getluadir{#1}\relax
6570       #2 TRT\relax
6571     \fi
6572   \fi}

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

6573 \def\bbl@thedir{0}
6574 \def\bbl@textdir#1{%
6575   \bbl@setluadir{text}\textdir{#1}%
6576   \chardef\bbl@thetextdir#1\relax
6577   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6578   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6579 \def\bbl@pardir#1{% Used twice
6580   \bbl@setluadir{par}\pardir{#1}%
6581   \chardef\bbl@thepardir#1\relax}
6582 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6583 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6584 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

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

6585 \ifnum\bbl@bidimode>\z@ % Any bidi=
6586   \def\bbl@insidemath{0}%
6587   \def\bbl@everymath{\def\bbl@insidemath{1}}
6588   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6589   \frozen@everymath\expandafter{%
6590     \expandafter\bbl@everymath\the\frozen@everymath}
6591   \frozen@everydisplay\expandafter{%
6592     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6593   \AtBeginDocument{
6594     \directlua{
6595       function Babel.math_box_dir(head)
6596         if not (token.get_macro('bbl@insidemath') == '0') then
6597           if Babel.hlist_has_bidi(head) then

```

```

6598         local d = node.new(node.id'dir')
6599         d.dir = '+TRT'
6600         node.insert_before(head, node.has_glyph(head), d)
6601         local inmath = false
6602         for item in node.traverse(head) do
6603             if item.id == 11 then
6604                 inmath = (item.subtype == 0)
6605             elseif not inmath then
6606                 node.set_attribute(item,
6607                     Babel.attr_dir, token.get_macro('bbl@thedir'))
6608             end
6609         end
6610     end
6611 end
6612 return head
6613 end
6614 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6615     "Babel.math_box_dir", 0)
6616 if Babel.unset_atdir then
6617     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6618         "Babel.unset_atdir")
6619     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6620         "Babel.unset_atdir")
6621 end
6622 }}%
6623 \fi

Experimental. Tentative name.

6624 \DeclareRobustCommand\localebox[1]{%
6625     {\def\bbl@insidemath{0}%
6626         \mbox{\foreignlanguage{\language}\{#1\}}}}

```

10.12 Layout

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

```

6627 \bbl@trace{Redefinitions for bidi layout}
6628 %
6629 << *More package options >> ≡
6630 \chardef\bbl@eqnpos\z@
6631 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6632 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6633 << /More package options >>
6634 %
6635 \ifnum\bbl@bidimode>\z@ % Any bidi=
6636     \matheqdirmode\@ne % A luatex primitive

```

```

6637 \let\bbl@eqnodir\relax
6638 \def\bbl@eqdel{()}
6639 \def\bbl@eqnum{%
6640   {\normalfont\normalcolor
6641     \expandafter\@firstoftwo\bbl@eqdel
6642     \theequation
6643     \expandafter\@secondoftwo\bbl@eqdel}}
6644 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6645 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6646 \def\bbl@eqno@flip#1{%
6647   \ifdim\predisplaysize=-\maxdimen
6648     \eqno
6649     \hb@xt@.01pt{%
6650       \hb@xt@\displaywidth{\hss#1\glet\bbl@upset\@currentlabel}}\hss}%
6651   \else
6652     \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6653   \fi
6654   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6655 \def\bbl@leqno@flip#1{%
6656   \ifdim\predisplaysize=-\maxdimen
6657     \leqno
6658     \hb@xt@.01pt{%
6659       \hss\hb@xt@\displaywidth{#1\glet\bbl@upset\@currentlabel}\hss}}%
6660   \else
6661     \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6662   \fi
6663   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6664 %
6665 \AtBeginDocument{%
6666   \ifx\bbl@noamsmath\relax\else
6667     \ifx\maketag@@@% undefined % Normal equation, eqnarray
6668       \AddToHook{env/equation/begin}{%
6669         \ifnum\bbl@thetextdir>z@
6670           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6671           \let\@eqnnum\bbl@eqnum
6672           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6673           \chardef\bbl@thetextdir\z@
6674           \bbl@add\normalfont{\bbl@eqnodir}%
6675           \ifcase\bbl@eqnpos
6676             \let\bbl@puteqno\bbl@eqno@flip
6677           \or
6678             \let\bbl@puteqno\bbl@leqno@flip
6679           \fi
6680         \fi}%
6681       \ifnum\bbl@eqnpos=\tw@% else
6682         \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6683       \fi
6684       \AddToHook{env/eqnarray/begin}{%
6685         \ifnum\bbl@thetextdir>z@
6686           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6687           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6688           \chardef\bbl@thetextdir\z@
6689           \bbl@add\normalfont{\bbl@eqnodir}%
6690           \ifnum\bbl@eqnpos=\@ne
6691             \def\@eqnnum{%
6692               \setbox\z@\hbox{\bbl@eqnum}%
6693               \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6694           \else
6695             \let\@eqnnum\bbl@eqnum
6696           \fi
6697         \fi}
6698       % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6699       \expandafter\bbl@sreplace\csname\endcsname{${$}\eqno\kern.001pt$}$}%

```

```

6700 \else % amstex
6701 \bbl@exp{% Hack to hide maybe undefined conditionals:
6702 \chardef\bbl@eqnpos=0%
6703 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6704 \ifnum\bbl@eqnpos=\@ne
6705 \let\bbl@ams@lap\hbox
6706 \else
6707 \let\bbl@ams@lap\llap
6708 \fi
6709 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6710 \bbl@sreplace\intertext@{\normalbaselines}%
6711 {\normalbaselines
6712 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6713 \ExplSyntaxOff
6714 \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6715 \ifx\bbl@ams@lap\hbox % leqno
6716 \def\bbl@ams@flip#1{%
6717 \hbox to 0.01pt{\hss\hbox to\displaywidth{#1}\hss}}%
6718 \else % eqno
6719 \def\bbl@ams@flip#1{%
6720 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6721 \fi
6722 \def\bbl@ams@preset#1{%
6723 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6724 \ifnum\bbl@thetextdir>\z@
6725 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6726 \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6727 \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6728 \fi}%
6729 \ifnum\bbl@eqnpos=\tw@\else
6730 \def\bbl@ams@equation{%
6731 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6732 \ifnum\bbl@thetextdir>\z@
6733 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6734 \chardef\bbl@thetextdir\z@
6735 \bbl@add\normalfont{\bbl@eqnodir}%
6736 \ifcase\bbl@eqnpos
6737 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6738 \or
6739 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6740 \fi
6741 \fi}%
6742 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6743 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6744 \fi
6745 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6746 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6747 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6748 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6749 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6750 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6751 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6752 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6753 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6754 % Hackish, for proper alignment. Don't ask me why it works!:
6755 \bbl@exp{% Avoid a 'visible' conditional
6756 \\\AddToHook{env/align*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6757 \\\AddToHook{env/alignat*/end}{\<iftag>\<else>\\tag*{\<fi>}}%
6758 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6759 \AddToHook{env/split/before}{%
6760 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6761 \ifnum\bbl@thetextdir>\z@
6762 \bbl@ifsamestring\@currentvir{equation}%

```

```

6763         {\ifx\bbbl@ams@lap\hbox % legno
6764         \def\bbbl@ams@flip#1{%
6765         \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6766         \else
6767         \def\bbbl@ams@flip#1{%
6768         \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6769         \fi}%
6770         {}%
6771         \fi}%
6772     \fi\fi}
6773 \fi

```

Declarations specific to lua, called by \babelprovide.

```

6774 \def\bbbl@provide@extra#1{%
6775     % == onchar ==
6776     \ifx\bbbl@KVP@onchar\@nnil\else
6777         \bbbl@luahyphenate
6778         \bbbl@exp{%
6779             \\\AddToHook{env/document/before}{%
6780             {%\let\\bbbl@ifrestoring\\@firstoftwo
6781             \\\select@language{#1}{}}}%
6782         \directlua{
6783             if Babel.locale_mapped == nil then
6784                 Babel.locale_mapped = true
6785                 Babel.linebreaking.add_before(Babel.locale_map, 1)
6786                 Babel.loc_to_scr = {}
6787                 Babel.chr_to_loc = Babel.chr_to_loc or {}
6788             end
6789             Babel.locale_props[\the\localeid].letters = false
6790         }%
6791         \bbbl@xin@{ letters }{ \bbbl@KVP@onchar\space}%
6792         \ifin@
6793             \directlua{
6794                 Babel.locale_props[\the\localeid].letters = true
6795             }%
6796         \fi
6797         \bbbl@xin@{ ids }{ \bbbl@KVP@onchar\space}%
6798         \ifin@
6799             \ifx\bbbl@starthyphens\@undefined % Needed if no explicit selection
6800                 \AddBabelHook{babel-onchar}{beforestart}{\bbbl@starthyphens}%
6801             \fi
6802             \bbbl@exp{\bbbl@add\bbbl@starthyphens
6803             {\bbbl@patterns@lua{\language\language}}}%
6804             \directlua{
6805                 if Babel.script_blocks['\bbbl@cl{sbc}'] then
6806                     Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbbl@cl{sbc}']
6807                     Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@language}\space
6808                 end
6809             }%
6810         \fi
6811         \bbbl@xin@{ fonts }{ \bbbl@KVP@onchar\space}%
6812         \ifin@
6813             \bbbl@ifunset{bbbl@lsys@language}{\bbbl@provide@lsys{language}}{%
6814             \bbbl@ifunset{bbbl@wdir@language}{\bbbl@provide@dirs{language}}{%
6815             \directlua{
6816                 if Babel.script_blocks['\bbbl@cl{sbc}'] then
6817                     Babel.loc_to_scr[\the\localeid] =
6818                     Babel.script_blocks['\bbbl@cl{sbc}']
6819                 end}%
6820             \ifx\bbbl@mapselect\@undefined
6821                 \AtBeginDocument{%
6822                     \bbbl@patchfont{\bbbl@mapselect}}%
6823                     {\selectfont}}%

```

```

6824 \def\bb@mapselect{%
6825 \let\bb@mapselect\relax
6826 \edef\bb@prefontid{\fontid\font}}%
6827 \def\bb@mapdir##1{%
6828 \begingroup
6829 \setbox\z@\hbox{% Force text mode
6830 \def\language{##1}%
6831 \let\bb@ifrestoring\@firstoftwo % To avoid font warning
6832 \bb@switchfont
6833 \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6834 \directlua{
6835 Babel.locale_props[\the\csname \bb@id@##1\endcsname]%
6836 [\bb@prefontid] = \fontid\font\space}%
6837 \fi}%
6838 \endgroup}%
6839 \fi
6840 \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\language}}}%
6841 \fi
6842 \fi
6843 % == mapfont ==
6844 % For bidi texts, to switch the font based on direction. Deprecated
6845 \ifx\bb@KVP@mapfont\@nnil\else
6846 \bb@ifsamestring{\bb@KVP@mapfont}{direction}}{%
6847 {\bb@error{unknown-mapfont}}}%
6848 \bb@ifunset{\bb@lsys\language}{\bb@provide\lsys{\language}}}%
6849 \bb@ifunset{\bb@wdir\language}{\bb@provide\dirs{\language}}}%
6850 \ifx\bb@mapselect\@undefined
6851 \AtBeginDocument{%
6852 \bb@patchfont{\bb@mapselect}}%
6853 {\selectfont}}%
6854 \def\bb@mapselect{%
6855 \let\bb@mapselect\relax
6856 \edef\bb@prefontid{\fontid\font}}%
6857 \def\bb@mapdir##1{%
6858 {\def\language{##1}%
6859 \let\bb@ifrestoring\@firstoftwo % avoid font warning
6860 \bb@switchfont
6861 \directlua{Babel.fontmap
6862 [\the\csname \bb@wdir@##1\endcsname]%
6863 [\bb@prefontid]=\fontid\font}}}%
6864 \fi
6865 \bb@exp{\bb@add\bb@mapselect{\bb@mapdir{\language}}}%
6866 \fi
6867 % == Line breaking: CJK quotes ==
6868 \ifcase\bb@engine\or
6869 \bb@xin{/c}{\bb@cl{\lnbrk}}%
6870 \ifin@
6871 \bb@ifunset{\bb@quote\language}}{%
6872 {\directlua{
6873 Babel.locale_props[\the\localeid].cjk_quotes = {}
6874 local cs = 'op'
6875 for c in string.utfvalues(
6876 [[\csname \bb@quote\language\endcsname]]) do
6877 if Babel.cjk_characters[c].c == 'qu' then
6878 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6879 end
6880 cs = ( cs == 'op') and 'cl' or 'op'
6881 end
6882 }}}%
6883 \fi
6884 \fi
6885 % == Counters: mapdigits ==
6886 % Native digits

```

```

6887 \ifx\bbl@KVP@mapdigits\@nnil\else
6888 \bbl@ifunset\bbl@dgnat@\languagename\}%
6889 {\bbl@activate@preotf
6890 \directlua{
6891   Babel.digits_mapped = true
6892   Babel.digits = Babel.digits or {}
6893   Babel.digits[\the\localeid] =
6894     table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6895   if not Babel.numbers then
6896     function Babel.numbers(head)
6897       local LOCALE = Babel.attr_locale
6898       local GLYPH = node.id'glyph'
6899       local inmath = false
6900       for item in node.traverse(head) do
6901         if not inmath and item.id == GLYPH then
6902           local temp = node.get_attribute(item, LOCALE)
6903           if Babel.digits[temp] then
6904             local chr = item.char
6905             if chr > 47 and chr < 58 then
6906               item.char = Babel.digits[temp][chr-47]
6907             end
6908           end
6909           elseif item.id == node.id'math' then
6910             inmath = (item.subtype == 0)
6911           end
6912         end
6913       return head
6914     end
6915   end
6916 }}%
6917 \fi
6918 % == transforms ==
6919 \ifx\bbl@KVP@transforms\@nnil\else
6920 \def\bbl@elt##1##2##3{%
6921   \in@{$transforms.}{##1}%
6922   \ifin@
6923     \def\bbl@tempa{##1}%
6924     \bbl@replace\bbl@tempa{transforms.}{}%
6925     \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6926   \fi}%
6927 \bbl@exp{%
6928   \\bbl@ifblank{\bbl@cl{dgnat}}}%
6929   {\let\\bbl@tempa\relax}%
6930   {\def\\bbl@tempa{%
6931     \\bbl@elt{transforms.prehyphenation}%
6932     {digits.native.1.0}{([0-9])}%
6933     \\bbl@elt{transforms.prehyphenation}%
6934     {digits.native.1.1}{string={1\string|0123456789\string|\bbl@cl{dgnat}}}}}%
6935 \ifx\bbl@tempa\relax\else
6936 \toks@{\expandafter\expandafter\expandafter{%
6937   \csname bbl@inidata@\languagename\endcsname}%
6938   \bbl@csarg\edef{inidata@\languagename}{%
6939     \unexpanded\expandafter{\bbl@tempa}%
6940     \the\toks@}%
6941 \fi
6942 \csname bbl@inidata@\languagename\endcsname
6943 \bbl@release@transforms\relax % \relax closes the last item.
6944 \fi}

```

Start tabular here:

```

6945 \def\localerestoredirs{%
6946   \ifcase\bbl@thetextdir
6947     \ifnum\textdirection=\z@\else\textdir TLT\fi

```

```

6948 \else
6949 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6950 \fi
6951 \ifcase\bbl@thepardir
6952 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6953 \else
6954 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6955 \fi}
6956 %
6957 \IfBabelLayout{tabular}%
6958 {\chardef\bbl@tabular@mode\tw}% All RTL
6959 {\IfBabelLayout{notabular}%
6960 {\chardef\bbl@tabular@mode\z}%
6961 {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6962 %
6963 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6964 % Redefine: vrules mess up dirs.
6965 \def\@arstrut{\relax\copy\@arstrutbox}%
6966 \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6967 \let\bbl@parabefore\relax
6968 \AddToHook{para/before}{\bbl@parabefore}
6969 \AtBeginDocument{%
6970 \bbl@replace\@tabular{$}{$%
6971 \def\bbl@insidemath{0}%
6972 \def\bbl@parabefore{\localerestoredirs}}%
6973 \ifnum\bbl@tabular@mode=\@ne
6974 \bbl@ifunset{\@tabclassz}{}%
6975 \bbl@exp{% Hide conditionals
6976 \\\bbl@sreplace\\\@tabclassz
6977 {\<ifcase>\\\@chnum}%
6978 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6979 \@ifpackageloaded{colortbl}%
6980 {\bbl@sreplace\@classz
6981 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6982 {\@ifpackageloaded{array}%
6983 {\bbl@exp{% Hide conditionals
6984 \\\bbl@sreplace\\\@classz
6985 {\<ifcase>\\\@chnum}%
6986 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6987 \\\bbl@sreplace\\\@classz
6988 {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
6989 {}}%
6990 \fi}%
6991 \or % 2 = All RTL - tabular
6992 \let\bbl@parabefore\relax
6993 \AddToHook{para/before}{\bbl@parabefore}
6994 \AtBeginDocument{%
6995 \@ifpackageloaded{colortbl}%
6996 {\bbl@replace\@tabular{$}{$%
6997 \def\bbl@insidemath{0}%
6998 \def\bbl@parabefore{\localerestoredirs}}%
6999 \bbl@sreplace\@classz
7000 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
7001 {}}%
7002 \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.

```

7003 \AtBeginDocument{%
7004 \@ifpackageloaded{multicol}%
7005 {\toks@\expandafter{\multi@column@out}%
7006 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%

```



```

7007     {}%
7008     \@ifpackageloaded{paracol}%
7009     {\edef\pcol@output{%
7010       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7011     {}}%
7012 \fi

```

Finish here if there in no layout.

```

7013 \ifx\bbl@opt@layout\@nnil\endinput\fi

```

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

```

7014 \ifnum\bbl@bidimode>\z@ % Any bidi=
7015   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
7016     \bbl@exp{%
7017       \mathdir\the\bodydir
7018       #1%           Once entered in math, set boxes to restore values
7019       \def\\bbl@insidemath{0}%
7020       \<ifmmode>%
7021       \everyvbox{%
7022         \the\everyvbox
7023         \bodydir\the\bodydir
7024         \mathdir\the\mathdir
7025         \everyhbox{\the\everyhbox}%
7026         \everyvbox{\the\everyvbox}}%
7027       \everyhbox{%
7028         \the\everyhbox
7029         \bodydir\the\bodydir
7030         \mathdir\the\mathdir
7031         \everyhbox{\the\everyhbox}%
7032         \everyvbox{\the\everyvbox}}%
7033       \<fi>}}%
7034 \IfBabelLayout{nopars}
7035 {}
7036 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7037 \IfBabelLayout{pars}
7038 {\def\@hangfrom#1{%
7039   \setbox\@tempboxa\hbox{#{#1}}%
7040   \hangindent\wd\@tempboxa
7041   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
7042     \shapemode\@ne
7043     \fi
7044   \noindent\box\@tempboxa}}
7045 {}
7046 \fi
7047 %
7048 \IfBabelLayout{tabular}
7049 {\let\bbl@0L@@tabular\@tabular
7050  \bbl@replace\@tabular{$}{\bbl@nextfake$}%
7051  \let\bbl@NL@@tabular\@tabular
7052  \AtBeginDocument{%
7053    \ifx\bbl@NL@@tabular\@tabular\else
7054      \bbl@exp{\in{\bbl@nextfake}{\@tabular}}}%
7055    \ifin\else
7056      \bbl@replace\@tabular{$}{\bbl@nextfake$}%
7057    \fi
7058    \let\bbl@NL@@tabular\@tabular
7059    \fi}}
7060 {}
7061 %
7062 \IfBabelLayout{lists}

```

```

7063 {\let\bbl@OL@list\list
7064 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
7065 \let\bbl@NL@list\list
7066 \def\bbl@listparshape#1#2#3{%
7067 \parshape #1 #2 #3 %
7068 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
7069 \shapemode\tw@
7070 \fi}}
7071 {}
7072 %
7073 \IfBabelLayout{graphics}
7074 {\let\bbl@pictresetdir\relax
7075 \def\bbl@pictsetdir#1{%
7076 \ifcase\bbl@thetextdir
7077 \let\bbl@pictresetdir\relax
7078 \else
7079 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7080 \or\textdir TLT
7081 \else\bodydir TLT \textdir TLT
7082 \fi
7083 % \(\text|par)dir required in pgf:
7084 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7085 \fi}%
7086 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
7087 \directlua{
7088 Babel.get_picture_dir = true
7089 Babel.picture_has_bidi = 0
7090 %
7091 function Babel.picture_dir (head)
7092 if not Babel.get_picture_dir then return head end
7093 if Babel.hlist_has_bidi(head) then
7094 Babel.picture_has_bidi = 1
7095 end
7096 return head
7097 end
7098 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7099 "Babel.picture_dir")
7100 }%
7101 \AtBeginDocument{%
7102 \def\LS@rot{%
7103 \setbox\@outputbox\vbox{%
7104 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
7105 \long\def\put(#1,#2)#3{%
7106 \@killglue
7107 % Try:
7108 \ifx\bbl@pictresetdir\relax
7109 \def\bbl@tempc{0}%
7110 \else
7111 \directlua{
7112 Babel.get_picture_dir = true
7113 Babel.picture_has_bidi = 0
7114 }%
7115 \setbox\z@\hb@xt@z@{%
7116 \@defaultunitsset\@tempdimc{#1}\unitlength
7117 \kern\@tempdimc
7118 #3\hss}%
7119 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7120 \fi
7121 % Do:
7122 \@defaultunitsset\@tempdimc{#2}\unitlength
7123 \raise\@tempdimc\hb@xt@z@{%
7124 \@defaultunitsset\@tempdimc{#1}\unitlength
7125 \kern\@tempdimc

```

```

7126      {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
7127      \ignorespaces}%
7128      \MakeRobust\put}%
7129  \AtBeginDocument
7130  {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
7131   \ifx\pgfpicture\@undefined\else
7132     \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
7133     \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
7134     \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
7135   \fi
7136   \ifx\tikzpicture\@undefined\else
7137     \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
7138     \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
7139     \bbbl@sreplace\tikz{\begingroup}\begingroup\bbbl@pictsetdir\tw@}%
7140     \bbbl@sreplace\tikzpicture{\begingroup}\begingroup\bbbl@pictsetdir\tw@}%
7141   \fi
7142   \ifx\tcolorbox\@undefined\else
7143     \def\tcb@drawing@env@begin{%
7144       \csname tcb@before@\tcb@split@state\endcsname
7145       \bbbl@pictsetdir\tw@
7146       \begin{\kv tcb@graphenv}%
7147       \tcb@bbdraw
7148       \tcb@apply@graph@patches}%
7149     \def\tcb@drawing@env@end{%
7150       \end{\kv tcb@graphenv}%
7151       \bbbl@pictresetdir
7152       \csname tcb@after@\tcb@split@state\endcsname}%
7153   \fi
7154   }}
7155   {}

```

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.

```

7156 \IfBabelLayout{counters*}%
7157 {\bbbl@add\bbbl@opt@layout{.counters.}%
7158  \directlua{
7159    luatexbase.add_to_callback("process_output_buffer",
7160      Babel.discard_sublr , "Babel.discard_sublr") }%
7161  }{}
7162 \IfBabelLayout{counters}%
7163 {\let\bbbl@0L@@textsuperscript\@textsuperscript
7164  \bbbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7165  \let\bbbl@latinarabic=\@arabic
7166  \let\bbbl@0L@@arabic\@arabic
7167  \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
7168  \ifpackagewith{babel}{bidi=default}%
7169    {\let\bbbl@asciroman=\@roman
7170     \let\bbbl@0L@@roman\@roman
7171     \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
7172     \let\bbbl@asciiRoman=\@Roman
7173     \let\bbbl@0L@@roman\@Roman
7174     \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
7175     \let\bbbl@0L@labelenumii\labelenumii
7176     \def\labelenumii{\theenumii}%
7177     \let\bbbl@0L@p@enumiii\p@enumiii
7178     \def\p@enumiii{\p@enumii}\theenumii{}}{}{}

```

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

```

7179 \IfBabelLayout{extras}%
7180 {\bbbl@ncarg\let\bbbl@0L@underline{underline }%
7181  \bbbl@carg\bbbl@sreplace{underline }%
7182  {\$@@@underline}{\bgroup\bbbl@nextfake$@@@underline}%

```

```

7183 \bbl@carg\bbl@sreplace{underline }%
7184 {\m@th$}{\m@th$\egroup}%
7185 \let\bbl@0L@LaTeXe\LaTeXe
7186 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7187 \if b\expandafter\@car\@f@series\@nil\boldmath\fi
7188 \babelsublr{%
7189 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
7190 {}
7191 \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.

```

7192 (*transforms
7193 Babel.linebreaking.replacements = {}
7194 Babel.linebreaking.replacements[0] = {} -- pre
7195 Babel.linebreaking.replacements[1] = {} -- post
7196
7197 function Babel.tovalue(v)
7198   if type(v) == 'table' then
7199     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7200   else
7201     return v
7202   end
7203 end
7204
7205 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7206
7207 function Babel.set_hboxed(head, gc)
7208   for item in node.traverse(head) do
7209     node.set_attribute(item, Babel.attr_hboxed, 1)
7210   end
7211   return head
7212 end
7213
7214 Babel.fetch_subtext = {}
7215
7216 Babel.ignore_pre_char = function(node)
7217   return (node.lang == Babel.nohyphenation)
7218 end
7219
7220 Babel.show_transforms = false
7221
7222 -- Merging both functions doesn't seem feasible, because there are too
7223 -- many differences.
7224 Babel.fetch_subtext[0] = function(head)
7225   local word_string = ''
7226   local word_nodes = {}
7227   local lang
7228   local item = head
7229   local inmath = false
7230
7231   while item do

```

```

7232
7233     if item.id == 11 then
7234         inmath = (item.subtype == 0)
7235     end
7236
7237     if inmath then
7238         -- pass
7239
7240     elseif item.id == 29 then
7241         local locale = node.get_attribute(item, Babel.attr_locale)
7242
7243         if lang == locale or lang == nil then
7244             lang = lang or locale
7245             if Babel.ignore_pre_char(item) then
7246                 word_string = word_string .. Babel.us_char
7247             else
7248                 if node.has_attribute(item, Babel.attr_hboxed) then
7249                     word_string = word_string .. Babel.us_char
7250                 else
7251                     word_string = word_string .. unicode.utf8.char(item.char)
7252                 end
7253             end
7254             word_nodes[#word_nodes+1] = item
7255         else
7256             break
7257         end
7258
7259     elseif item.id == 12 and item.subtype == 13 then
7260         if node.has_attribute(item, Babel.attr_hboxed) then
7261             word_string = word_string .. Babel.us_char
7262         else
7263             word_string = word_string .. ' '
7264         end
7265         word_nodes[#word_nodes+1] = item
7266
7267         -- Ignore leading unrecognized nodes, too.
7268     elseif word_string ~= '' then
7269         word_string = word_string .. Babel.us_char
7270         word_nodes[#word_nodes+1] = item -- Will be ignored
7271     end
7272
7273     item = item.next
7274 end
7275
7276 -- Here and above we remove some trailing chars but not the
7277 -- corresponding nodes. But they aren't accessed.
7278 if word_string:sub(-1) == ' ' then
7279     word_string = word_string:sub(1,-2)
7280 end
7281 if Babel.show_transforms then texio.write_nl(word_string) end
7282 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7283 return word_string, word_nodes, item, lang
7284 end
7285
7286 Babel.fetch_subtext[1] = function(head)
7287     local word_string = ''
7288     local word_nodes = {}
7289     local lang
7290     local item = head
7291     local inmath = false
7292
7293     while item do
7294

```

```

7295     if item.id == 11 then
7296         inmath = (item.subtype == 0)
7297     end
7298
7299     if inmath then
7300         -- pass
7301
7302     elseif item.id == 29 then
7303         if item.lang == lang or lang == nil then
7304             lang = lang or item.lang
7305             if node.has_attribute(item, Babel.attr_hboxed) then
7306                 word_string = word_string .. Babel.us_char
7307             elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7308                 word_string = word_string .. Babel.us_char
7309             else
7310                 word_string = word_string .. unicode.utf8.char(item.char)
7311             end
7312             word_nodes[#word_nodes+1] = item
7313         else
7314             break
7315         end
7316
7317     elseif item.id == 7 and item.subtype == 2 then
7318         if node.has_attribute(item, Babel.attr_hboxed) then
7319             word_string = word_string .. Babel.us_char
7320         else
7321             word_string = word_string .. '='
7322         end
7323         word_nodes[#word_nodes+1] = item
7324
7325     elseif item.id == 7 and item.subtype == 3 then
7326         if node.has_attribute(item, Babel.attr_hboxed) then
7327             word_string = word_string .. Babel.us_char
7328         else
7329             word_string = word_string .. '|'
7330         end
7331         word_nodes[#word_nodes+1] = item
7332
7333     -- (1) Go to next word if nothing was found, and (2) implicitly
7334     -- remove leading USs.
7335     elseif word_string == '' then
7336         -- pass
7337
7338     -- This is the responsible for splitting by words.
7339     elseif (item.id == 12 and item.subtype == 13) then
7340         break
7341
7342     else
7343         word_string = word_string .. Babel.us_char
7344         word_nodes[#word_nodes+1] = item -- Will be ignored
7345     end
7346
7347     item = item.next
7348 end
7349 if Babel.show_transforms then texio.write_nl(word_string) end
7350 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7351 return word_string, word_nodes, item, lang
7352 end
7353
7354 function Babel.pre_hyphenate_replace(head)
7355     Babel.hyphenate_replace(head, 0)
7356 end
7357

```

```

7358 function Babel.post_hyphenate_replace(head)
7359   Babel.hyphenate_replace(head, 1)
7360 end
7361
7362 Babel.us_char = string.char(31)
7363
7364 function Babel.hyphenate_replace(head, mode)
7365   local u = unicode.utf8
7366   local lbkr = Babel.linebreaking.replacements[mode]
7367   local tovalue = Babel.tovalue
7368
7369   local word_head = head
7370
7371   if Babel.show_transforms then
7372     texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7373   end
7374
7375   while true do -- for each subtext block
7376
7377     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7378
7379     if Babel.debug then
7380       print()
7381       print((mode == 0) and '@@@@<' or '@@@@>', w)
7382     end
7383
7384     if nw == nil and w == '' then break end
7385
7386     if not lang then goto next end
7387     if not lbkr[lang] then goto next end
7388
7389     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7390     -- loops are nested.
7391     for k=1, #lbkr[lang] do
7392       local p = lbkr[lang][k].pattern
7393       local r = lbkr[lang][k].replace
7394       local attr = lbkr[lang][k].attr or -1
7395
7396       if Babel.debug then
7397         print('*****', p, mode)
7398       end
7399
7400       -- This variable is set in some cases below to the first *byte*
7401       -- after the match, either as found by u.match (faster) or the
7402       -- computed position based on sc if w has changed.
7403       local last_match = 0
7404       local step = 0
7405
7406       -- For every match.
7407       while true do
7408         if Babel.debug then
7409           print('====')
7410         end
7411         local new -- used when inserting and removing nodes
7412         local dummy_node -- used by after
7413
7414         local matches = { u.match(w, p, last_match) }
7415
7416         if #matches < 2 then break end
7417
7418         -- Get and remove empty captures (with ())'s, which return a
7419         -- number with the position), and keep actual captures
7420         -- (from (...)), if any, in matches.

```

```

7421     local first = table.remove(matches, 1)
7422     local last  = table.remove(matches, #matches)
7423     -- Non re-fetched substrings may contain \31, which separates
7424     -- subsubstrings.
7425     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7426
7427     local save_last = last -- with A()BC()D, points to D
7428
7429     -- Fix offsets, from bytes to unicode. Explained above.
7430     first = u.len(w:sub(1, first-1)) + 1
7431     last  = u.len(w:sub(1, last-1)) -- now last points to C
7432
7433     -- This loop stores in a small table the nodes
7434     -- corresponding to the pattern. Used by 'data' to provide a
7435     -- predictable behavior with 'insert' (w_nodes is modified on
7436     -- the fly), and also access to 'remove'd nodes.
7437     local sc = first-1 -- Used below, too
7438     local data_nodes = {}
7439
7440     local enabled = true
7441     for q = 1, last-first+1 do
7442         data_nodes[q] = w_nodes[sc+q]
7443         if enabled
7444             and attr > -1
7445             and not node.has_attribute(data_nodes[q], attr)
7446         then
7447             enabled = false
7448         end
7449     end
7450
7451     -- This loop traverses the matched substring and takes the
7452     -- corresponding action stored in the replacement list.
7453     -- sc = the position in substr nodes / string
7454     -- rc = the replacement table index
7455     local rc = 0
7456
7457     ----- TODO. dummy_node?
7458     while rc < last-first+1 or dummy_node do -- for each replacement
7459         if Babel.debug then
7460             print('.....', rc + 1)
7461         end
7462         sc = sc + 1
7463         rc = rc + 1
7464
7465         if Babel.debug then
7466             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7467             local ss = ''
7468             for itt in node.traverse(head) do
7469                 if itt.id == 29 then
7470                     ss = ss .. unicode.utf8.char(itt.char)
7471                 else
7472                     ss = ss .. '{' .. itt.id .. '}'
7473                 end
7474             end
7475             print('*****', ss)
7476
7477         end
7478
7479         local crep = r[rc]
7480         local item = w_nodes[sc]
7481         local item_base = item
7482         local placeholder = Babel.us_char
7483         local d

```



```

7484
7485     if crep and crep.data then
7486         item_base = data_nodes[crep.data]
7487     end
7488
7489     if crep then
7490         step = crep.step or step
7491     end
7492
7493     if crep and crep.after then
7494         crep.insert = true
7495         if dummy_node then
7496             item = dummy_node
7497         else -- TODO. if there is a node after?
7498             d = node.copy(item_base)
7499             head, item = node.insert_after(head, item, d)
7500             dummy_node = item
7501         end
7502     end
7503
7504     if crep and not crep.after and dummy_node then
7505         node.remove(head, dummy_node)
7506         dummy_node = nil
7507     end
7508
7509     if not enabled then
7510         last_match = save_last
7511         goto next
7512
7513     elseif crep and next(crep) == nil then -- = {}
7514         if step == 0 then
7515             last_match = save_last -- Optimization
7516         else
7517             last_match = utf8.offset(w, sc+step)
7518         end
7519         goto next
7520
7521     elseif crep == nil or crep.remove then
7522         node.remove(head, item)
7523         table.remove(w_nodes, sc)
7524         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7525         sc = sc - 1 -- Nothing has been inserted.
7526         last_match = utf8.offset(w, sc+1+step)
7527         goto next
7528
7529     elseif crep and crep.kashida then -- Experimental
7530         node.set_attribute(item,
7531             Babel.attr_kashida,
7532             crep.kashida)
7533         last_match = utf8.offset(w, sc+1+step)
7534         goto next
7535
7536     elseif crep and crep.string then
7537         local str = crep.string(matches)
7538         if str == '' then -- Gather with nil
7539             node.remove(head, item)
7540             table.remove(w_nodes, sc)
7541             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7542             sc = sc - 1 -- Nothing has been inserted.
7543         else
7544             local loop_first = true
7545             for s in string.utfvalues(str) do
7546                 d = node.copy(item_base)

```

```

7547         d.char = s
7548         if loop_first then
7549             loop_first = false
7550             head, new = node.insert_before(head, item, d)
7551             if sc == 1 then
7552                 word_head = head
7553             end
7554             w_nodes[sc] = d
7555             w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7556         else
7557             sc = sc + 1
7558             head, new = node.insert_before(head, item, d)
7559             table.insert(w_nodes, sc, new)
7560             w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7561         end
7562         if Babel.debug then
7563             print('.....', 'str')
7564             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7565         end
7566         end -- for
7567         node.remove(head, item)
7568     end -- if ''
7569     last_match = utf8.offset(w, sc+1+step)
7570     goto next
7571
7572 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7573     d = node.new(7, 3) -- (disc, regular)
7574     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7575     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7576     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7577     d.attr = item_base.attr
7578     if crep.pre == nil then -- TeXbook p96
7579         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7580     else
7581         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7582     end
7583     placeholder = '|'
7584     head, new = node.insert_before(head, item, d)
7585
7586 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7587     -- ERROR
7588
7589 elseif crep and crep.penalty then
7590     d = node.new(14, 0) -- (penalty, userpenalty)
7591     d.attr = item_base.attr
7592     d.penalty = tovalue(crep.penalty)
7593     head, new = node.insert_before(head, item, d)
7594
7595 elseif crep and crep.space then
7596     -- 655360 = 10 pt = 10 * 65536 sp
7597     d = node.new(12, 13) -- (glue, spaceskip)
7598     local quad = font.getfont(item_base.font).size or 655360
7599     node.setglue(d, tovalue(crep.space[1]) * quad,
7600                 tovalue(crep.space[2]) * quad,
7601                 tovalue(crep.space[3]) * quad)
7602     if mode == 0 then
7603         placeholder = ' '
7604     end
7605     head, new = node.insert_before(head, item, d)
7606
7607 elseif crep and crep.norule then
7608     -- 655360 = 10 pt = 10 * 65536 sp
7609     d = node.new(2, 3) -- (rule, empty) = \no*rule

```

```

7610     local quad = font.getfont(item_base.font).size or 655360
7611     d.width  = tovalue(crep.norule[1]) * quad
7612     d.height = tovalue(crep.norule[2]) * quad
7613     d.depth  = tovalue(crep.norule[3]) * quad
7614     head, new = node.insert_before(head, item, d)
7615
7616 elseif crep and crep.spacefactor then
7617     d = node.new(12, 13)      -- (glue, spaceskip)
7618     local base_font = font.getfont(item_base.font)
7619     node.setglue(d,
7620         tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7621         tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7622         tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7623     if mode == 0 then
7624         placeholder = ' '
7625     end
7626     head, new = node.insert_before(head, item, d)
7627
7628 elseif mode == 0 and crep and crep.space then
7629     -- ERROR
7630
7631 elseif crep and crep.kern then
7632     d = node.new(13, 1)      -- (kern, user)
7633     local quad = font.getfont(item_base.font).size or 655360
7634     d.attr = item_base.attr
7635     d.kern = tovalue(crep.kern) * quad
7636     head, new = node.insert_before(head, item, d)
7637
7638 elseif crep and crep.node then
7639     d = node.new(crep.node[1], crep.node[2])
7640     d.attr = item_base.attr
7641     head, new = node.insert_before(head, item, d)
7642
7643 end -- i.e., replacement cases
7644
7645 -- Shared by disc, space(factor), kern, node and penalty.
7646 if sc == 1 then
7647     word_head = head
7648 end
7649 if crep.insert then
7650     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7651     table.insert(w_nodes, sc, new)
7652     last = last + 1
7653 else
7654     w_nodes[sc] = d
7655     node.remove(head, item)
7656     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7657 end
7658
7659 last_match = utf8.offset(w, sc+1+step)
7660
7661 ::next::
7662
7663 end -- for each replacement
7664
7665 if Babel.show_transforms then texio.write_nl('> ' .. w) end
7666 if Babel.debug then
7667     print('.....', '/')
7668     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7669 end
7670
7671 if dummy_node then
7672     node.remove(head, dummy_node)

```

```

7673     dummy_node = nil
7674     end
7675
7676     end -- for match
7677
7678     end -- for patterns
7679
7680     ::next::
7681     word_head = nw
7682 end -- for substring
7683
7684 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7685 return head
7686 end
7687
7688 -- This table stores capture maps, numbered consecutively
7689 Babel.capture_maps = {}
7690
7691 function Babel.esc_hex_to_char(h)
7692   if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7693     tex.getcatcode(tonumber(h, 16)) ~= 12 then
7694     return string.format([[Uchar"%X ]], tonumber(h,16))
7695   else
7696     return unicode.utf8.char(tonumber(h, 16))
7697   end
7698 end
7699
7700 -- The following functions belong to the next macro
7701 function Babel.capture_func(key, cap)
7702   local ret = "[[" .. cap:gsub('{{([0-9])}}', "[ ]..m[%1]..[[") .. "]"
7703   local cnt
7704   local u = unicode.utf8
7705   ret = u.gsub(ret, '{{(%x%x%x%x+)}', '\x01%\x04')
7706   ret, cnt = ret:gsub('{{([0-9])|([^\^]|+)|(.-)}', Babel.capture_func_map)
7707   ret = u.gsub(ret, '\x01(%x%x%x%x+)\x04', Babel.esc_hex_to_char)
7708   ret = ret:gsub("%[%[%]%]%.%", '')
7709   ret = ret:gsub("%.%[%[%]%]%", '')
7710   return key .. [[=function(m) return ]] .. ret .. [[ end]]
7711 end
7712
7713 function Babel.capt_map(from, mapno)
7714   return Babel.capture_maps[mapno][from] or from
7715 end
7716
7717 -- Handle the {n|abc|ABC} syntax in captures
7718 function Babel.capture_func_map(capno, from, to)
7719   local u = unicode.utf8
7720   from = u.gsub(from, '\x01(%x%x%x%x+)\x04',
7721     function (n)
7722       return u.char(tonumber(n, 16))
7723     end)
7724   to = u.gsub(to, '\x01(%x%x%x%x+)\x04',
7725     function (n)
7726       return u.char(tonumber(n, 16))
7727     end)
7728   local froms = {}
7729   for s in string.utfcharacters(from) do
7730     table.insert(froms, s)
7731   end
7732   local cnt = 1
7733   table.insert(Babel.capture_maps, {})
7734   local mlen = table.getn(Babel.capture_maps)
7735   for s in string.utfcharacters(to) do

```

```

7736     Babel.capture_maps[mlen][froms[cnt]] = s
7737     cnt = cnt + 1
7738 end
7739 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7740     (mlen) .. " ).." .. "[["
7741 end
7742
7743 -- Create/Extend reversed sorted list of kashida weights:
7744 function Babel.capture_kashida(key, wt)
7745     wt = tonumber(wt)
7746     if Babel.kashida_wts then
7747         for p, q in ipairs(Babel.kashida_wts) do
7748             if wt == q then
7749                 break
7750             elseif wt > q then
7751                 table.insert(Babel.kashida_wts, p, wt)
7752                 break
7753             elseif table.getn(Babel.kashida_wts) == p then
7754                 table.insert(Babel.kashida_wts, wt)
7755             end
7756         end
7757     else
7758         Babel.kashida_wts = { wt }
7759     end
7760     return 'kashida = ' .. wt
7761 end
7762
7763 function Babel.capture_node(id, subtype)
7764     local sbt = 0
7765     for k, v in pairs(node.subtypes(id)) do
7766         if v == subtype then sbt = k end
7767     end
7768     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7769 end
7770
7771 -- Experimental: applies prehyphenation transforms to a string (letters
7772 -- and spaces).
7773 function Babel.string_prehyphenation(str, locale)
7774     local n, head, last, res
7775     head = node.new(8, 0) -- dummy (hack just to start)
7776     last = head
7777     for s in string.utfvalues(str) do
7778         if s == 20 then
7779             n = node.new(12, 0)
7780         else
7781             n = node.new(29, 0)
7782             n.char = s
7783         end
7784         node.set_attribute(n, Babel.attr_locale, locale)
7785         last.next = n
7786         last = n
7787     end
7788     head = Babel.hyphenate_replace(head, 0)
7789     res = ''
7790     for n in node.traverse(head) do
7791         if n.id == 12 then
7792             res = res .. ' '
7793         elseif n.id == 29 then
7794             res = res .. unicode.utf8.char(n.char)
7795         end
7796     end
7797     tex.print(res)
7798 end

```

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.

```
7800 ⟨\*basic-r⟩
7801 Babel.bidi_enabled = true
7802
7803 require('babel-data-bidi.lua')
7804
7805 local characters = Babel.characters
7806 local ranges = Babel.ranges
7807
7808 local DIR = node.id("dir")
7809
7810 local function dir_mark(head, from, to, outer)
7811   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7812   local d = node.new(DIR)
7813   d.dir = '+' .. dir
7814   node.insert_before(head, from, d)
7815   d = node.new(DIR)
7816   d.dir = '-' .. dir
7817   node.insert_after(head, to, d)
7818 end
7819
7820 function Babel.bidi(head, ispar)
7821   local first_n, last_n          -- first and last char with nums
```

```

7822 local last_es          -- an auxiliary 'last' used with nums
7823 local first_d, last_d   -- first and last char in L/R block
7824 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):

```

7825 local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7826 local strong_lr = (strong == 'l') and 'l' or 'r'
7827 local outer = strong
7828
7829 local new_dir = false
7830 local first_dir = false
7831 local inmath = false
7832
7833 local last_lr
7834
7835 local type_n = ''
7836
7837 for item in node.traverse(head) do
7838
7839   -- three cases: glyph, dir, otherwise
7840   if item.id == node.id'glyph'
7841     or (item.id == 7 and item.subtype == 2) then
7842
7843     local itemchar
7844     if item.id == 7 and item.subtype == 2 then
7845       itemchar = item.replace.char
7846     else
7847       itemchar = item.char
7848     end
7849     local chardata = characters[itemchar]
7850     dir = chardata and chardata.d or nil
7851     if not dir then
7852       for nn, et in ipairs(ranges) do
7853         if itemchar < et[1] then
7854           break
7855         elseif itemchar <= et[2] then
7856           dir = et[3]
7857           break
7858         end
7859       end
7860     end
7861     dir = dir or 'l'
7862     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).

```

7863   if new_dir then
7864     attr_dir = 0
7865     for at in node.traverse(item.attr) do
7866       if at.number == Babel.attr_dir then
7867         attr_dir = at.value & 0x3
7868       end
7869     end
7870     if attr_dir == 1 then
7871       strong = 'r'
7872     elseif attr_dir == 2 then
7873       strong = 'al'
7874     else
7875       strong = 'l'

```

```

7876     end
7877     strong_lr = (strong == 'l') and 'l' or 'r'
7878     outer = strong_lr
7879     new_dir = false
7880 end
7881
7882 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

7883     dir_real = dir          -- We need dir_real to set strong below
7884     if dir == 'al' then dir = 'r' end -- W3

```

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

```

7885     if strong == 'al' then
7886         if dir == 'en' then dir = 'an' end          -- W2
7887         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7888         strong_lr = 'r'                             -- W3
7889     end

```

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

```

7890     elseif item.id == node.id'dir' and not inmath then
7891         new_dir = true
7892         dir = nil
7893     elseif item.id == node.id'math' then
7894         inmath = (item.subtype == 0)
7895     else
7896         dir = nil          -- Not a char
7897     end

```

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

```

7898     if dir == 'en' or dir == 'an' or dir == 'et' then
7899         if dir ~= 'et' then
7900             type_n = dir
7901         end
7902         first_n = first_n or item
7903         last_n = last_es or item
7904         last_es = nil
7905     elseif dir == 'es' and last_n then -- W3+W6
7906         last_es = item
7907     elseif dir == 'cs' then          -- it's right - do nothing
7908     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7909         if strong_lr == 'r' and type_n ~= '' then
7910             dir_mark(head, first_n, last_n, 'r')
7911         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7912             dir_mark(head, first_n, last_n, 'r')
7913             dir_mark(head, first_d, last_d, outer)
7914             first_d, last_d = nil, nil
7915         elseif strong_lr == 'l' and type_n ~= '' then
7916             last_d = last_n
7917         end
7918         type_n = ''
7919         first_n, last_n = nil, nil
7920     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:

```

7921     if dir == 'l' or dir == 'r' then

```



```

7922     if dir ~= outer then
7923         first_d = first_d or item
7924         last_d = item
7925     elseif first_d and dir ~= strong_lr then
7926         dir_mark(head, first_d, last_d, outer)
7927         first_d, last_d = nil, nil
7928     end
7929 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.

```

7930     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7931         item.char = characters[item.char] and
7932             characters[item.char].m or item.char
7933     elseif (dir or new_dir) and last_lr ~= item then
7934         local mir = outer .. strong_lr .. (dir or outer)
7935         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7936             for ch in node.traverse(node.next(last_lr)) do
7937                 if ch == item then break end
7938                 if ch.id == node.id'glyph' and characters[ch.char] then
7939                     ch.char = characters[ch.char].m or ch.char
7940                 end
7941             end
7942         end
7943     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).

```

7944     if dir == 'l' or dir == 'r' then
7945         last_lr = item
7946         strong = dir_real -- Don't search back - best save now
7947         strong_lr = (strong == 'l') and 'l' or 'r'
7948     elseif new_dir then
7949         last_lr = nil
7950     end
7951 end

```

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

```

7952     if last_lr and outer == 'r' then
7953         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7954             if characters[ch.char] then
7955                 ch.char = characters[ch.char].m or ch.char
7956             end
7957         end
7958     end
7959     if first_n then
7960         dir_mark(head, first_n, last_n, outer)
7961     end
7962     if first_d then
7963         dir_mark(head, first_d, last_d, outer)
7964     end

```

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

```

7965     return node.prev(head) or head
7966 end
7967 </basic-r>

```

And here the Lua code for bidi=basic:

```

7968 <*>basic<
7969 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>

```

```

7970
7971 Babel.fontmap = Babel.fontmap or {}
7972 Babel.fontmap[0] = {}      -- l
7973 Babel.fontmap[1] = {}      -- r
7974 Babel.fontmap[2] = {}      -- al/an
7975
7976 -- To cancel mirroring. Also OML, OMS, U?
7977 Babel.symbol_fonts = Babel.symbol_fonts or {}
7978 Babel.symbol_fonts[font.id('tenln')] = true
7979 Babel.symbol_fonts[font.id('tenlnw')] = true
7980 Babel.symbol_fonts[font.id('tencirc')] = true
7981 Babel.symbol_fonts[font.id('tencircw')] = true
7982
7983 Babel.bidi_enabled = true
7984 Babel.mirroring_enabled = true
7985
7986 require('babel-data-bidi.lua')
7987
7988 local characters = Babel.characters
7989 local ranges = Babel.ranges
7990
7991 local DIR = node.id('dir')
7992 local GLYPH = node.id('glyph')
7993
7994 local function insert_implicit(head, state, outer)
7995     local new_state = state
7996     if state.sim and state.eim and state.sim ~= state.eim then
7997         dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7998         local d = node.new(DIR)
7999         d.dir = '+' .. dir
8000         node.insert_before(head, state.sim, d)
8001         local d = node.new(DIR)
8002         d.dir = '-' .. dir
8003         node.insert_after(head, state.eim, d)
8004     end
8005     new_state.sim, new_state.eim = nil, nil
8006     return head, new_state
8007 end
8008
8009 local function insert_numeric(head, state)
8010     local new
8011     local new_state = state
8012     if state.san and state.ean and state.san ~= state.ean then
8013         local d = node.new(DIR)
8014         d.dir = '+TLT'
8015         _, new = node.insert_before(head, state.san, d)
8016         if state.san == state.sim then state.sim = new end
8017         local d = node.new(DIR)
8018         d.dir = '-TLT'
8019         _, new = node.insert_after(head, state.ean, d)
8020         if state.ean == state.eim then state.eim = new end
8021     end
8022     new_state.san, new_state.ean = nil, nil
8023     return head, new_state
8024 end
8025
8026 local function glyph_not_symbol_font(node)
8027     if node.id == GLYPH then
8028         return not Babel.symbol_fonts[node.font]
8029     else
8030         return false
8031     end
8032 end

```

```

8033
8034 -- TODO - \hbox with an explicit dir can lead to wrong results
8035 -- <R \hbox dir TLT{<R>} and <L \hbox dir TRT{<L>}>. A small attempt
8036 -- was made to improve the situation, but the problem is the 3-dir
8037 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8038 -- well.
8039
8040 function Babel.bidi(head, ispar, hdir)
8041   local d    -- d is used mainly for computations in a loop
8042   local prev_d = ''
8043   local new_d = false
8044
8045   local nodes = {}
8046   local outer_first = nil
8047   local inmath = false
8048
8049   local glue_d = nil
8050   local glue_i = nil
8051
8052   local has_en = false
8053   local first_et = nil
8054
8055   local has_hyperlink = false
8056
8057   local ATDIR = Babel.attr_dir
8058   local attr_d, temp
8059   local locale_d
8060
8061   local save_outer
8062   local locale_d = node.get_attribute(head, ATDIR)
8063   if locale_d then
8064     locale_d = locale_d & 0x3
8065     save_outer = (locale_d == 0 and 'l') or
8066                  (locale_d == 1 and 'r') or
8067                  (locale_d == 2 and 'al')
8068   elseif ispar then      -- Or error? Shouldn't happen
8069     -- when the callback is called, we are just _after_ the box,
8070     -- and the textdir is that of the surrounding text
8071     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8072   else                  -- Empty box
8073     save_outer = ('TRT' == hdir) and 'r' or 'l'
8074   end
8075   local outer = save_outer
8076   local last = outer
8077   -- 'al' is only taken into account in the first, current loop
8078   if save_outer == 'al' then save_outer = 'r' end
8079
8080   local fontmap = Babel.fontmap
8081
8082   for item in node.traverse(head) do
8083
8084     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8085     locale_d = node.get_attribute(item, ATDIR)
8086     node.set_attribute(item, ATDIR, 0x80)
8087
8088     -- In what follows, #node is the last (previous) node, because the
8089     -- current one is not added until we start processing the neutrals.
8090     -- three cases: glyph, dir, otherwise
8091     if glyph_not_symbol_font(item)
8092       or (item.id == 7 and item.subtype == 2) then
8093
8094       if locale_d == 0x80 then goto nextnode end
8095

```

```

8096     local d_font = nil
8097     local item_r
8098     if item.id == 7 and item.subtype == 2 then
8099         item_r = item.replace -- automatic discs have just 1 glyph
8100     else
8101         item_r = item
8102     end
8103
8104     local chardata = characters[item_r.char]
8105     d = chardata and chardata.d or nil
8106     if not d or d == 'nsm' then
8107         for nn, et in ipairs(ranges) do
8108             if item_r.char < et[1] then
8109                 break
8110             elseif item_r.char <= et[2] then
8111                 if not d then d = et[3]
8112                 elseif d == 'nsm' then d_font = et[3]
8113                 end
8114                 break
8115             end
8116         end
8117     end
8118     d = d or 'l'
8119
8120     -- A short 'pause' in bidi for mapfont
8121     -- %%% TODO. move if fontmap here
8122     d_font = d_font or d
8123     d_font = (d_font == 'l' and 0) or
8124             (d_font == 'nsm' and 0) or
8125             (d_font == 'r' and 1) or
8126             (d_font == 'al' and 2) or
8127             (d_font == 'an' and 2) or nil
8128     if d_font and fontmap and fontmap[d_font][item_r.font] then
8129         item_r.font = fontmap[d_font][item_r.font]
8130     end
8131
8132     if new_d then
8133         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8134         if inmath then
8135             attr_d = 0
8136         else
8137             attr_d = locale_d & 0x3
8138         end
8139         if attr_d == 1 then
8140             outer_first = 'r'
8141             last = 'r'
8142         elseif attr_d == 2 then
8143             outer_first = 'r'
8144             last = 'al'
8145         else
8146             outer_first = 'l'
8147             last = 'l'
8148         end
8149         outer = last
8150         has_en = false
8151         first_et = nil
8152         new_d = false
8153     end
8154
8155     if glue_d then
8156         if (d == 'l' and 'l' or 'r') ~= glue_d then
8157             table.insert(nodes, {glue_i, 'on', nil})
8158         end

```

```

8159         glue_d = nil
8160         glue_i = nil
8161     end
8162
8163     elseif item.id == DIR then
8164         d = nil
8165         new_d = true
8166
8167     elseif item.id == node.id'glue' and item.subtype == 13 then
8168         glue_d = d
8169         glue_i = item
8170         d = nil
8171
8172     elseif item.id == node.id'math' then
8173         inmath = (item.subtype == 0)
8174
8175     elseif item.id == 8 and item.subtype == 19 then
8176         has_hyperlink = true
8177
8178     else
8179         d = nil
8180     end
8181
8182     -- AL <= EN/ET/ES      -- W2 + W3 + W6
8183     if last == 'al' and d == 'en' then
8184         d = 'an'          -- W3
8185     elseif last == 'al' and (d == 'et' or d == 'es') then
8186         d = 'on'          -- W6
8187     end
8188
8189     -- EN + CS/ES + EN      -- W4
8190     if d == 'en' and #nodes >= 2 then
8191         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8192             and nodes[#nodes-1][2] == 'en' then
8193             nodes[#nodes][2] = 'en'
8194         end
8195     end
8196
8197     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8198     if d == 'an' and #nodes >= 2 then
8199         if (nodes[#nodes][2] == 'cs')
8200             and nodes[#nodes-1][2] == 'an' then
8201             nodes[#nodes][2] = 'an'
8202         end
8203     end
8204
8205     -- ET/EN                -- W5 + W7->l / W6->on
8206     if d == 'et' then
8207         first_et = first_et or (#nodes + 1)
8208     elseif d == 'en' then
8209         has_en = true
8210         first_et = first_et or (#nodes + 1)
8211     elseif first_et then      -- d may be nil here !
8212         if has_en then
8213             if last == 'l' then
8214                 temp = 'l'    -- W7
8215             else
8216                 temp = 'en'   -- W5
8217             end
8218         else
8219             temp = 'on'       -- W6
8220         end
8221         for e = first_et, #nodes do

```

```

8222         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8223     end
8224     first_et = nil
8225     has_en = false
8226 end
8227
8228 -- Force mathdir in math if ON (currently works as expected only
8229 -- with 'l')
8230
8231 if inmath and d == 'on' then
8232     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8233 end
8234
8235 if d then
8236     if d == 'al' then
8237         d = 'r'
8238         last = 'al'
8239     elseif d == 'l' or d == 'r' then
8240         last = d
8241     end
8242     prev_d = d
8243     table.insert(nodes, {item, d, outer_first})
8244 end
8245
8246 outer_first = nil
8247
8248 ::nextnode::
8249
8250 end -- for each node
8251
8252 -- TODO -- repeated here in case EN/ET is the last node. Find a
8253 -- better way of doing things:
8254 if first_et then -- dir may be nil here !
8255     if has_en then
8256         if last == 'l' then
8257             temp = 'l' -- W7
8258         else
8259             temp = 'en' -- W5
8260         end
8261     else
8262         temp = 'on' -- W6
8263     end
8264     for e = first_et, #nodes do
8265         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8266     end
8267 end
8268
8269 -- dummy node, to close things
8270 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8271
8272 ----- NEUTRAL -----
8273
8274 outer = save_outer
8275 last = outer
8276
8277 local first_on = nil
8278
8279 for q = 1, #nodes do
8280     local item
8281
8282     local outer_first = nodes[q][3]
8283     outer = outer_first or outer
8284     last = outer_first or last

```

```

8285
8286     local d = nodes[q][2]
8287     if d == 'an' or d == 'en' then d = 'r' end
8288     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8289
8290     if d == 'on' then
8291         first_on = first_on or q
8292     elseif first_on then
8293         if last == d then
8294             temp = d
8295         else
8296             temp = outer
8297         end
8298         for r = first_on, q - 1 do
8299             nodes[r][2] = temp
8300             item = nodes[r][1] -- MIRRORING
8301             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8302                 and temp == 'r' and characters[item.char] then
8303                 local font_mode = ''
8304                 if item.font > 0 and font.fonts[item.font].properties then
8305                     font_mode = font.fonts[item.font].properties.mode
8306                 end
8307                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8308                     item.char = characters[item.char].m or item.char
8309                 end
8310             end
8311         end
8312         first_on = nil
8313     end
8314
8315     if d == 'r' or d == 'l' then last = d end
8316 end
8317
8318 ----- IMPLICIT, REORDER -----
8319
8320 outer = save_outer
8321 last = outer
8322
8323 local state = {}
8324 state.has_r = false
8325
8326 for q = 1, #nodes do
8327
8328     local item = nodes[q][1]
8329
8330     outer = nodes[q][3] or outer
8331
8332     local d = nodes[q][2]
8333
8334     if d == 'nsm' then d = last end -- W1
8335     if d == 'en' then d = 'an' end
8336     local isdir = (d == 'r' or d == 'l')
8337
8338     if outer == 'l' and d == 'an' then
8339         state.san = state.san or item
8340         state.ean = item
8341     elseif state.san then
8342         head, state = insert_numeric(head, state)
8343     end
8344
8345     if outer == 'l' then
8346         if d == 'an' or d == 'r' then -- im -> implicit
8347             if d == 'r' then state.has_r = true end

```

```

8348     state.sim = state.sim or item
8349     state.eim = item
8350     elseif d == 'l' and state.sim and state.has_r then
8351         head, state = insert_implicit(head, state, outer)
8352     elseif d == 'l' then
8353         state.sim, state.eim, state.has_r = nil, nil, false
8354     end
8355 else
8356     if d == 'an' or d == 'l' then
8357         if nodes[q][3] then -- nil except after an explicit dir
8358             state.sim = item -- so we move sim 'inside' the group
8359         else
8360             state.sim = state.sim or item
8361         end
8362         state.eim = item
8363     elseif d == 'r' and state.sim then
8364         head, state = insert_implicit(head, state, outer)
8365     elseif d == 'r' then
8366         state.sim, state.eim = nil, nil
8367     end
8368 end
8369
8370 if isdir then
8371     last = d -- Don't search back - best save now
8372 elseif d == 'on' and state.san then
8373     state.san = state.san or item
8374     state.ean = item
8375 end
8376
8377 end
8378
8379 head = node.prev(head) or head
8380 % \end{macrocode}
8381 %
8382 % Now direction nodes has been distributed with relation to characters
8383 % and spaces, we need to take into account \TeX-specific elements in
8384 % the node list, to move them at an appropriate place. Firstly, with
8385 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8386 % that the latter are still discardable.
8387 %
8388 % \begin{macrocode}
8389 --- FIXES ---
8390 if has_hyperlink then
8391     local flag, linking = 0, 0
8392     for item in node.traverse(head) do
8393         if item.id == DIR then
8394             if item.dir == '+TRT' or item.dir == '+TLT' then
8395                 flag = flag + 1
8396             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8397                 flag = flag - 1
8398             end
8399         elseif item.id == 8 and item.subtype == 19 then
8400             linking = flag
8401         elseif item.id == 8 and item.subtype == 20 then
8402             if linking > 0 then
8403                 if item.prev.id == DIR and
8404                     (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8405                     d = node.new(DIR)
8406                     d.dir = item.prev.dir
8407                     node.remove(head, item.prev)
8408                     node.insert_after(head, item, d)
8409                 end
8410             end

```



```

8411         linking = 0
8412     end
8413 end
8414 end
8415
8416 for item in node.traverse_id(10, head) do
8417     local p = item
8418     local flag = false
8419     while p.prev and p.prev.id == 14 do
8420         flag = true
8421         p = p.prev
8422     end
8423     if flag then
8424         node.insert_before(head, p, node.copy(item))
8425         node.remove(head,item)
8426     end
8427 end
8428
8429 return head
8430 end
8431
8432 function Babel.unset_atdir(head)
8433     local ATDIR = Babel.attr_dir
8434     for item in node.traverse(head) do
8435         node.set_attribute(item, ATDIR, 0x80)
8436     end
8437 return head
8438 end
8439 </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.

```

8439 <{*nil>
8440 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8441 \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.

```

8442 \ifx\l@nil\undefined
8443     \newlanguage\l@nil
8444     \namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8445     \let\bbl@elt\relax
8446     \edef\bbl@languages{% Add it to the list of languages

```

```

8447 \bbl@languages\bbl@elt{nil}{\the\l@nil}{}}
8448 \fi

```

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

```

8449 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

\captionnil

\datenil

```

8450 \let\captionnil\@empty
8451 \let\datenil\@empty

```

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

```

8452 \def\bbl@inidata@nil{%
8453 \bbl@elt{identification}{tag.ini}{und}%
8454 \bbl@elt{identification}{load.level}{0}%
8455 \bbl@elt{identification}{charset}{utf8}%
8456 \bbl@elt{identification}{version}{1.0}%
8457 \bbl@elt{identification}{date}{2022-05-16}%
8458 \bbl@elt{identification}{name.local}{nil}%
8459 \bbl@elt{identification}{name.english}{nil}%
8460 \bbl@elt{identification}{name.babel}{nil}%
8461 \bbl@elt{identification}{tag.bcp47}{und}%
8462 \bbl@elt{identification}{language.tag.bcp47}{und}%
8463 \bbl@elt{identification}{tag.opentype}{dflt}%
8464 \bbl@elt{identification}{script.name}{Latin}%
8465 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8466 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8467 \bbl@elt{identification}{level}{1}%
8468 \bbl@elt{identification}{encodings}{}%
8469 \bbl@elt{identification}{derivate}{no}}
8470 \@namedef{bbl@tbc@nil}{und}
8471 \@namedef{bbl@lbc@nil}{und}
8472 \@namedef{bbl@casing@nil}{und}
8473 \@namedef{bbl@lotf@nil}{dflt}
8474 \@namedef{bbl@elname@nil}{nil}
8475 \@namedef{bbl@lname@nil}{nil}
8476 \@namedef{bbl@esname@nil}{Latin}
8477 \@namedef{bbl@sname@nil}{Latin}
8478 \@namedef{bbl@sbc@nil}{Latn}
8479 \@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.

```

8480 \ldf@finish{nil}
8481 \</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.

```

8482 \<Compute Julian day> ≡
8483 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8484 \def\bbl@cs@gregleap#1{%
8485 (\bbl@fpmmod{#1}{4} == 0) &&
8486 (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8487 \def\bbl@cs@jd#1#2#3{% year, month, day
8488 \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8489 floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +

```

```

8490 floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
8491 ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
8492 <</Compute Julian day>>

```

13.1. Islamic

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

```

8493 <*ca-islamic>
8494 <@Compute Julian day>
8495 % == islamic (default)
8496 % Not yet implemented
8497 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{
    The Civil calendar.
8498 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8499 ((#3 + ceil(29.5 * (#2 - 1)) +
8500 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8501 1948439.5) - 1) }
8502 \namedef\bbl@ca@islamic-civil++{\bbl@ca@islamicvl@x{+2}}
8503 \namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl@x{+1}}
8504 \namedef\bbl@ca@islamic-civil{\bbl@ca@islamicvl@x{}}
8505 \namedef\bbl@ca@islamic-civil-{\bbl@ca@islamicvl@x{-1}}
8506 \namedef\bbl@ca@islamic-civil--{\bbl@ca@islamicvl@x{-2}}
8507 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8508 \edef\bbl@tempa{%
8509 \fpeval{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8510 \edef#5{%
8511 \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8512 \edef#6{\fpeval{
8513 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8514 \edef#7{\fpeval{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}

```

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

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

```

8515 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8516 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8517 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8518 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8519 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8520 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8521 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8522 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8523 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8524 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8525 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8526 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8527 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8528 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8529 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8530 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8531 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8532 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8533 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8534 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8535 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8536 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8537 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8538 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8539 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8540 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8541 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%

```

```

8542 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8543 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8544 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8545 65401,65431,65460,65490,65520}
8546 \namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8547 \namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8548 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8549 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8550 \ifnum#2>2014 \ifnum#2<2038
8551 \bbl@afterfi\expandafter\@gobble
8552 \fi\fi
8553 {\bbl@error{year-out-range}{2014-2038}{}}}%
8554 \edef\bbl@tempd{\fpeval{ % (Julian) day
8555 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8556 \count@\@ne
8557 \bbl@foreach\bbl@cs@umalqura@data{%
8558 \advance\count@\@ne
8559 \ifnum##1>\bbl@tempd\else
8560 \edef\bbl@tempe{\the\count@}%
8561 \edef\bbl@tempb{##1}%
8562 \fi}%
8563 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8564 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1 ) / 12) }}% annus
8565 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8566 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8567 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8568 \bbl@add\bbl@precalendar{%
8569 \bbl@replace\bbl@ld@calendar{-civil}}}%
8570 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8571 \bbl@replace\bbl@ld@calendar{+}}}%
8572 \bbl@replace\bbl@ld@calendar{-}}}%
8573 </ca-islamic

```

13.2. Hebrew

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

```

8574 < *ca-hebrew
8575 \newcount\bbl@cntcommon
8576 \def\bbl@remainder#1#2#3{%
8577 #3=#1\relax
8578 \divide #3 by #2\relax
8579 \multiply #3 by -#2\relax
8580 \advance #3 by #1\relax}%
8581 \newif\ifbbl@divisible
8582 \def\bbl@checkifdivisible#1#2{%
8583 {\countdef\tmp=0
8584 \bbl@remainder{#1}{#2}{\tmp}%
8585 \ifnum \tmp=0
8586 \global\bbl@divisibletrue
8587 \else
8588 \global\bbl@divisiblefalse
8589 \fi}}
8590 \newif\ifbbl@gregleap
8591 \def\bbl@ifgregleap#1{%
8592 \bbl@checkifdivisible{#1}{4}%
8593 \ifbbl@divisible
8594 \bbl@checkifdivisible{#1}{100}%
8595 \ifbbl@divisible
8596 \bbl@checkifdivisible{#1}{400}%
8597 \ifbbl@divisible
8598 \bbl@gregleaptrue

```

```

8599         \else
8600             \bbl@gregleapfalse
8601         \fi
8602     \else
8603         \bbl@gregleaptrue
8604     \fi
8605 \else
8606     \bbl@gregleapfalse
8607 \fi
8608 \ifbbl@gregleap}
8609 \def\bbl@gregdayspriormonths#1#2#3{%
8610     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8611         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8612     \bbl@ifgregleap{#2}%
8613     \ifnum #1 > 2
8614         \advance #3 by 1
8615     \fi
8616     \fi
8617     \global\bbl@cntcommon=#3}%
8618     #3=\bbl@cntcommon}
8619 \def\bbl@gregdaysprioryears#1#2{%
8620     {\countdef\tmpc=4
8621     \countdef\tmpb=2
8622     \tmpb=#1\relax
8623     \advance \tmpb by -1
8624     \tmpc=\tmpb
8625     \multiply \tmpc by 365
8626     #2=\tmpc
8627     \tmpc=\tmpb
8628     \divide \tmpc by 4
8629     \advance #2 by \tmpc
8630     \tmpc=\tmpb
8631     \divide \tmpc by 100
8632     \advance #2 by -\tmpc
8633     \tmpc=\tmpb
8634     \divide \tmpc by 400
8635     \advance #2 by \tmpc
8636     \global\bbl@cntcommon=#2\relax}%
8637     #2=\bbl@cntcommon}
8638 \def\bbl@absfromgreg#1#2#3#4{%
8639     {\countdef\tmpd=0
8640     #4=#1\relax
8641     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8642     \advance #4 by \tmpd
8643     \bbl@gregdaysprioryears{#3}{\tmpd}%
8644     \advance #4 by \tmpd
8645     \global\bbl@cntcommon=#4\relax}%
8646     #4=\bbl@cntcommon}
8647 \newif\ifbbl@hebrleap
8648 \def\bbl@checkleaphebryear#1{%
8649     {\countdef\tmpa=0
8650     \countdef\tmpb=1
8651     \tmpa=#1\relax
8652     \multiply \tmpa by 7
8653     \advance \tmpa by 1
8654     \bbl@remainder{\tmpa}{19}{\tmpb}%
8655     \ifnum \tmpb < 7
8656         \global\bbl@hebrleaptrue
8657     \else
8658         \global\bbl@hebrleapfalse
8659     \fi}}
8660 \def\bbl@hebrlapsedmonths#1#2{%
8661     {\countdef\tmpa=0

```

```

8662 \countdef\tmpb=1
8663 \countdef\tmpc=2
8664 \tmpa=#1\relax
8665 \advance \tmpa by -1
8666 #2=\tmpa
8667 \divide #2 by 19
8668 \multiply #2 by 235
8669 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8670 \tmpc=\tmpb
8671 \multiply \tmpb by 12
8672 \advance #2 by \tmpb
8673 \multiply \tmpc by 7
8674 \advance \tmpc by 1
8675 \divide \tmpc by 19
8676 \advance #2 by \tmpc
8677 \global\bbl@cntcommon=#2}%
8678 #2=\bbl@cntcommon}
8679 \def\bbl@hebreleapseddays#1#2{%
8680 {\countdef\tmpa=0
8681 \countdef\tmpb=1
8682 \countdef\tmpc=2
8683 \bbl@hebreleapsedmonths{#1}{#2}%
8684 \tmpa=#2\relax
8685 \multiply \tmpa by 13753
8686 \advance \tmpa by 5604
8687 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8688 \divide \tmpa by 25920
8689 \multiply #2 by 29
8690 \advance #2 by 1
8691 \advance #2 by \tmpa
8692 \bbl@remainder{#2}{7}{\tmpa}%
8693 \ifnum \tmpc < 19440
8694 \ifnum \tmpc < 9924
8695 \else
8696 \ifnum \tmpa=2
8697 \bbl@checkleaphebrewyear{#1}% of a common year
8698 \ifbbl@hebrleap
8699 \else
8700 \advance #2 by 1
8701 \fi
8702 \fi
8703 \fi
8704 \ifnum \tmpc < 16789
8705 \else
8706 \ifnum \tmpa=1
8707 \advance #1 by -1
8708 \bbl@checkleaphebrewyear{#1}% at the end of leap year
8709 \ifbbl@hebrleap
8710 \advance #2 by 1
8711 \fi
8712 \fi
8713 \fi
8714 \else
8715 \advance #2 by 1
8716 \fi
8717 \bbl@remainder{#2}{7}{\tmpa}%
8718 \ifnum \tmpa=0
8719 \advance #2 by 1
8720 \else
8721 \ifnum \tmpa=3
8722 \advance #2 by 1
8723 \else
8724 \ifnum \tmpa=5

```

```

8725         \advance #2 by 1
8726     \fi
8727 \fi
8728 \fi
8729 \global\bbl@cntcommon=#2\relax}%
8730 #2=\bbl@cntcommon}
8731 \def\bbl@daysinhebrewyear#1#2{%
8732     {\countdef\tmpe=12
8733     \bbl@hebreleapseddays{#1}{\tmpe}%
8734     \advance #1 by 1
8735     \bbl@hebreleapseddays{#1}{#2}%
8736     \advance #2 by -\tmpe
8737     \global\bbl@cntcommon=#2}%
8738     #2=\bbl@cntcommon}
8739 \def\bbl@hebrdayspriormonths#1#2#3{%
8740     {\countdef\tmpf= 14
8741     #3=\ifcase #1
8742         0 \or
8743         0 \or
8744         30 \or
8745         59 \or
8746         89 \or
8747         118 \or
8748         148 \or
8749         148 \or
8750         177 \or
8751         207 \or
8752         236 \or
8753         266 \or
8754         295 \or
8755         325 \or
8756         400
8757     \fi
8758     \bbl@checkleaphebrewyear{#2}%
8759     \ifbbl@hebrleap
8760         \ifnum #1 > 6
8761             \advance #3 by 30
8762         \fi
8763     \fi
8764     \bbl@daysinhebrewyear{#2}{\tmpf}%
8765     \ifnum #1 > 3
8766         \ifnum \tmpf=353
8767             \advance #3 by -1
8768         \fi
8769         \ifnum \tmpf=383
8770             \advance #3 by -1
8771         \fi
8772     \fi
8773     \ifnum #1 > 2
8774         \ifnum \tmpf=355
8775             \advance #3 by 1
8776         \fi
8777         \ifnum \tmpf=385
8778             \advance #3 by 1
8779         \fi
8780     \fi
8781     \global\bbl@cntcommon=#3\relax}%
8782     #3=\bbl@cntcommon}
8783 \def\bbl@absfromhebr#1#2#3#4{%
8784     {#4=#1\relax
8785     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8786     \advance #4 by #1\relax
8787     \bbl@hebreleapseddays{#3}{#1}%

```

```

8788 \advance #4 by #1\relax
8789 \advance #4 by -1373429
8790 \global\bbl@cntcommon=#4\relax}%
8791 #4=\bbl@cntcommon}
8792 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8793 {\countdef\tmpx= 17
8794 \countdef\tmpy= 18
8795 \countdef\tmpz= 19
8796 #6=#3\relax
8797 \global\advance #6 by 3761
8798 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8799 \tmpz=1 \tmpy=1
8800 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8801 \ifnum \tmpx > #4\relax
8802 \global\advance #6 by -1
8803 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8804 \fi
8805 \advance #4 by -\tmpx
8806 \advance #4 by 1
8807 #5=#4\relax
8808 \divide #5 by 30
8809 \loop
8810 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8811 \ifnum \tmpx < #4\relax
8812 \advance #5 by 1
8813 \tmpy=\tmpx
8814 \repeat
8815 \global\advance #5 by -1
8816 \global\advance #4 by -\tmpy}}
8817 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8818 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8819 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8820 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8821 \bbl@hebrfromgreg
8822 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8823 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8824 \edef#4{\the\bbl@hebryear}%
8825 \edef#5{\the\bbl@hebrmonth}%
8826 \edef#6{\the\bbl@hebrday}}
8827 </ca-hebrew

```

13.3. Persian

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

```

8828 <*ca-persian
8829 <@Compute Julian day@>
8830 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8831 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8832 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8833 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8834 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8835 \bbl@afterfi\expandafter\@gobble
8836 \fi\fi
8837 {\bbl@error{year-out-range}{2013-2050}{}}}%
8838 \bbl@xin{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8839 \ifin\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8840 \edef\bbl@tempc{\fpeval{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8841 \edef\bbl@tempb{\fpeval{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8842 \ifnum\bbl@tempc<\bbl@tempb

```



```

8843 \edef\bbl@tempa{\fpeval{\bbl@tempa-1}}% go back 1 year and redo
8844 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8845 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8846 \edef\bbl@tempb{\fpeval{\bbl@cs@jdd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8847 \fi
8848 \edef#4{\fpeval{\bbl@tempa-621}}% set Jalali year
8849 \edef#6{\fpeval{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8850 \edef#5{\fpeval{% set Jalali month
8851   (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8852 \edef#6{\fpeval{% set Jalali day
8853   (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
8854 </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.

```

8855 <*ca-coptic>
8856 <@Compute Julian day@>
8857 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8858 \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jdd{#1}{#2}{#3}) + 0.5}}%
8859 \edef\bbl@tempc{\fpeval{\bbl@tempd - 1825029.5}}%
8860 \edef#4{\fpeval{%
8861   floor((\bbl@tempc - floor(\bbl@tempc+366) / 1461)) / 365) + 1}}%
8862 \edef\bbl@tempc{\fpeval{%
8863   \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8864 \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
8865 \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}
8866 </ca-coptic>
8867 <*ca-ethiopic>
8868 <@Compute Julian day@>
8869 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8870 \edef\bbl@tempd{\fpeval{floor(\bbl@cs@jdd{#1}{#2}{#3}) + 0.5}}%
8871 \edef\bbl@tempc{\fpeval{\bbl@tempd - 1724220.5}}%
8872 \edef#4{\fpeval{%
8873   floor((\bbl@tempc - floor(\bbl@tempc+366) / 1461)) / 365) + 1}}%
8874 \edef\bbl@tempc{\fpeval{%
8875   \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8876 \edef#5{\fpeval{floor(\bbl@tempc / 30) + 1}}%
8877 \edef#6{\fpeval{\bbl@tempc - (#5 - 1) * 30 + 1}}
8878 </ca-ethiopic>

```

13.5. Julian

```

8879 <*ca-julian>
8880 <@Compute Julian day@>
8881 \def\bbl@ca@julian#1-#2-#3\@@#4#5#6{%
8882 \edef\bbl@tempj{\fpeval{floor(\bbl@cs@jdd{#1}{#2}{#3}) + .5}}%
8883 \edef\bbl@tempa{\fpeval{\bbl@tempj + 32082.5}}%
8884 \edef\bbl@tempb{\fpeval{floor((4 * \bbl@tempa + 3) / 1461)}}%
8885 \edef\bbl@tempc{\fpeval{\bbl@tempa - floor(1461*\bbl@tempb/4)}}%
8886 \edef\bbl@tempd{\fpeval{floor((5 * \bbl@tempc + 2) / 153)}}%
8887 \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
8888 \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
8889 \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}
8890 </ca-julian>

```

13.6. Buddhist

That's very simple.

```

8891 <*ca-buddhist>
8892 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8893 \edef#4{\number\numexpr#1+543\relax}%

```

```

8894 \edef#5{#2}%
8895 \edef#6{#3}}
8896 </ca-buddhist
8897 %
8898 % \subsection{Chinese}
8899 %
8900 % Brute force, with the Julian day of first day of each month. The
8901 % table has been computed with the help of \textsf{python-lunardate} by
8902 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8903 % is 2015-2044.
8904 %
8905 % \begin{macrocode}
8906 <{*ca-chinese
8907 \ExplSyntaxOn
8908 <@Compute Julian day@>
8909 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
8910 \edef\bbl@tempd{\fpeval{%
8911 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8912 \count@ \z@
8913 \@tempcnta=2015
8914 \bbl@foreach\bbl@cs@chinese@data{%
8915 \ifnum##1>\bbl@tempd\else
8916 \advance\count@\@ne
8917 \ifnum\count@>12
8918 \count@\@ne
8919 \advance\@tempcnta\@ne\fi
8920 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8921 \ifin@
8922 \advance\count@\m@ne
8923 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8924 \else
8925 \edef\bbl@tempe{\the\count@}%
8926 \fi
8927 \edef\bbl@tempb{##1}%
8928 \fi}%
8929 \edef#4{\the\@tempcnta}%
8930 \edef#5{\bbl@tempe}%
8931 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8932 \def\bbl@cs@chinese@leap{%
8933 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8934 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8935 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8936 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8937 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8938 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8939 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8940 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8941 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8942 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8943 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8944 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8945 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8946 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8947 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8948 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8949 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8950 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8951 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8952 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8953 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8954 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8955 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8956 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%

```

```

8957 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8958 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8959 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8960 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8961 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8962 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8963 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8964 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8965 10896,10926,10956,10986,11015,11045,11074,11103}
8966 \ExplSyntaxOff
8967 </ca-chinese>

```

14. Support for Plain T_EX (plain.def)

14.1. Not renaming hyphen.tex

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

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

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

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

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

```

8968 <*\bplain | blplain>
8969 \catcode\{=1 % left brace is begin-group character
8970 \catcode\}=2 % right brace is end-group character
8971 \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).

```

8972 \openin 0 hyphen.cfg
8973 \ifeof0
8974 \else
8975   \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.

```

8976 \def\input #1 {%
8977   \let\input\input
8978   \a hyphen.cfg
8979   \let\input\input
8980 }
8981 \fi
8982 </bplain | blplain>

```

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

```

8983 <bplain>\a plain.tex
8984 <blplain>\a lplain.tex

```

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

```

8985 <bplain>\def\fmtname{babel-plain}
8986 <blplain>\def\fmtname{babel-lplain}

```

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

14.2. Emulating some \LaTeX features

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

```
8987 <<{*Emulate LaTeX} >> ≡
8988 \def\@empty{}
8989 \def\loadlocalcfg#1{%
8990   \openin0#1.cfg
8991   \ifeof0
8992     \closein0
8993   \else
8994     \closein0
8995     {\immediate\write16{*****}%
8996       \immediate\write16{* Local config file #1.cfg used}%
8997       \immediate\write16{**}%
8998     }
8999     \input #1.cfg\relax
9000   \fi
9001   \@endofldf}
```

14.3. General tools

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

```
9002 \long\def\@firstofone#1{#1}
9003 \long\def\@firstoftwo#1#2{#1}
9004 \long\def\@secondoftwo#1#2{#2}
9005 \def\@nnil{\@nil}
9006 \def\@gobbletwo#1#2{}
9007 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
9008 \def\@staror@long#1{%
9009   \@ifstar
9010   {\let\l@ngrel@x\relax#1}%
9011   {\let\l@ngrel@x\long#1}}
9012 \let\l@ngrel@x\relax
9013 \def\@car#1#2\@nil{#1}
9014 \def\@cdr#1#2\@nil{#2}
9015 \let\@typeset@protect\relax
9016 \let\protected@edef\edef
9017 \long\def\@gobble#1{}
9018 \edef\@backslashchar{\expandafter\@gobble\string\}
9019 \def\strip@prefix#1>{}
9020 \def\g@addto@macro#1#2{{%
9021   \toks@\expandafter{#1#2}%
9022   \xdef#1{\the\toks@}}}
9023 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
9024 \def\@nameuse#1{\csname #1\endcsname}
9025 \def\@ifundefined#1{%
9026   \expandafter\ifx\csname#1\endcsname\relax
9027     \expandafter\@firstoftwo
9028   \else
9029     \expandafter\@secondoftwo
9030   \fi}
9031 \def\@expandtwoargs#1#2#3{%
9032   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9033 \def\zap@space#1 #2{%
9034   #1%
9035   \ifx#2\@empty\else\expandafter\zap@space\fi
9036   #2}
9037 \let\bbl@trace\@gobble
9038 \def\bbl@error#1{% Implicit #2#3#4}
```

```

9039 \begingroup
9040   \catcode`\=0   \catcode`\==12 \catcode`\`=12
9041   \catcode`\^M=5 \catcode`\%=14
9042   \input errbabel.def
9043 \endgroup
9044 \bbl@error{#1}}
9045 \def\bbl@warning#1{%
9046   \begingroup
9047     \newlinechar=`^^J
9048     \def\{`^^J(babel) }%
9049     \message{\#1}%
9050   \endgroup}
9051 \let\bbl@infowarn\bbl@warning
9052 \def\bbl@info#1{%
9053   \begingroup
9054     \newlinechar=`^^J
9055     \def\{`^^J}%
9056     \wlog{#1}%
9057   \endgroup}

```

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

```

9058 \ifx\@preamblecmds\undefined
9059   \def\@preamblecmds{}
9060 \fi
9061 \def\@onlypreamble#1{%
9062   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9063     \@preamblecmds\do#1}}
9064 \@onlypreamble\@onlypreamble

```

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

```

9065 \def\begindocument{%
9066   \@begindocumenthook
9067   \global\let\@begindocumenthook\undefined
9068   \def\do##1{\global\let##1\undefined}%
9069   \@preamblecmds
9070   \global\let\do\noexpand}
9071 \ifx\@begindocumenthook\undefined
9072   \def\@begindocumenthook{}
9073 \fi
9074 \@onlypreamble\@begindocumenthook
9075 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

9076 \def\AtEndOfPackage#1{\g@addto@macro\@endoflfd{#1}}
9077 \@onlypreamble\AtEndOfPackage
9078 \def\@endoflfd{}
9079 \@onlypreamble\@endoflfd
9080 \let\bbl@afterlang\@empty
9081 \chardef\bbl@opt@hyphenmap\z@

```

\LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer `\ifx`. The same trick is applied below.

```

9082 \catcode`\&=\z@
9083 \ifx@if@filesw\undefined
9084   \expandafter\let\csname if@filesw\expandafter\endcsname
9085     \csname iffalse\endcsname
9086 \fi
9087 \catcode`\&=4

```

Mimic \LaTeX 's commands to define control sequences.

```

9088 \def\newcommand{\@star@or@long\new@command}
9089 \def\new@command#1{%
9090   \@testopt{\@newcommand#1}0}
9091 \def\@newcommand#1[#2]{%
9092   \@ifnextchar [{\@xargdef#1[#2]}%
9093     {\@argdef#1[#2]}}
9094 \long\def\@argdef#1[#2]#3{%
9095   \@yargdef#1\@ne{#2}{#3}}
9096 \long\def\@xargdef#1[#2][#3]#4{%
9097   \expandafter\def\expandafter#1\expandafter{%
9098     \expandafter\@protected@testopt\expandafter #1%
9099     \csname\string#1\expandafter\endcsname{#3}}}%
9100 \expandafter\@yargdef \csname\string#1\endcsname
9101 \tw@{#2}{#4}}
9102 \long\def\@yargdef#1#2#3{%
9103   \@tempcnta#3\relax
9104   \advance \@tempcnta \@ne
9105   \let\@hash@\relax
9106   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9107   \@tempcntb #2%
9108   \@whilenum \@tempcntb < \@tempcnta
9109   \do{%
9110     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
9111     \advance\@tempcntb \@ne}%
9112   \let\@hash@###%
9113   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9114 \def\providecommand{\@star@or@long\provide@command}
9115 \def\provide@command#1{%
9116   \begingroup
9117     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9118   \endgroup
9119   \expandafter\@ifundefined\@gtempa
9120     {\def\reserved@a{\new@command#1}}%
9121     {\let\reserved@a\relax
9122     \def\reserved@a{\new@command\reserved@a}}%
9123   \reserved@a}%

9124 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9125 \def\declare@robustcommand#1{%
9126   \edef\reserved@a{\string#1}%
9127   \def\reserved@b{#1}%
9128   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9129   \edef#1{%
9130     \ifx\reserved@a\reserved@b
9131       \noexpand\x@protect
9132       \noexpand#1%
9133     \fi
9134     \noexpand\protect
9135     \expandafter\noexpand\csname
9136       \expandafter\@gobble\string#1 \endcsname
9137   }%
9138   \expandafter\new@command\csname
9139     \expandafter\@gobble\string#1 \endcsname
9140 }
9141 \def\x@protect#1{%
9142   \ifx\protect\@typeset@protect\else
9143     \@x@protect#1%
9144   \fi
9145 }
9146 \catcode`\&=\z@ % Trick to hide conditionals
9147 \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`.

```

9148 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9149 \catcode`\&=4
9150 \ifx\in@\@undefined
9151 \def\in@#1#2{%
9152 \def\in@@##1#1##2##3\in@@{%
9153 \ifx\in@@##2\in@false\else\in@true\fi}%
9154 \in@@##2#1\in@\in@@}
9155 \else
9156 \let\bbl@tempa\@empty
9157 \fi
9158 \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).

```

9159 \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.

```

9160 \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.

```

9161 \ifx\@tempcnta\@undefined
9162 \csname newcount\endcsname\@tempcnta\relax
9163 \fi
9164 \ifx\@tempcntb\@undefined
9165 \csname newcount\endcsname\@tempcntb\relax
9166 \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`).

```

9167 \ifx\bye\@undefined
9168 \advance\count10 by -2\relax
9169 \fi
9170 \ifx\@ifnextchar\@undefined
9171 \def\@ifnextchar#1#2#3{%
9172 \let\reserved@d=#1%
9173 \def\reserved@a{#2}\def\reserved@b{#3}%
9174 \futurelet\@let@token\@ifnch}
9175 \def\@ifnch{%
9176 \ifx\@let@token\@sptoken
9177 \let\reserved@c\@xifnch
9178 \else
9179 \ifx\@let@token\reserved@d
9180 \let\reserved@c\reserved@a
9181 \else
9182 \let\reserved@c\reserved@b
9183 \fi
9184 \fi
9185 \reserved@c}
9186 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
9187 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
9188 \fi
9189 \def\@testopt#1#2{%
9190 \@ifnextchar[#{1}{#1[#{2}]}}
9191 \def\@protected@testopt#1{%
9192 \ifx\protect\@typeset@protect
9193 \expandafter\@testopt

```

```

9194 \else
9195 \@@protect#1%
9196 \fi}
9197 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9198 #2\relax}\fi}
9199 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9200 \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9201 \def\DeclareTextCommand{%
9202 \@@dec@text@cmd\providecommand
9203 }
9204 \def\ProvideTextCommand{%
9205 \@@dec@text@cmd\providecommand
9206 }
9207 \def\DeclareTextSymbol#1#2#3{%
9208 \@@dec@text@cmd\chardef#1{#2}#3\relax
9209 }
9210 \def\@dec@text@cmd#1#2#3{%
9211 \expandafter\def\expandafter#2%
9212 \expandafter{%
9213 \csname#3-cmd\expandafter\endcsname
9214 \expandafter#2%
9215 \csname#3\string#2\endcsname
9216 }%
9217 % \let\@ifdefinable\@rc@ifdefinable
9218 \expandafter#1\csname#3\string#2\endcsname
9219 }
9220 \def\@current@cmd#1{%
9221 \ifx\protect\@typeset@protect\else
9222 \noexpand#1\expandafter\@gobble
9223 \fi
9224 }
9225 \def\@changed@cmd#1#2{%
9226 \ifx\protect\@typeset@protect
9227 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9228 \expandafter\ifx\csname ?\string#1\endcsname\relax
9229 \expandafter\def\csname ?\string#1\endcsname{%
9230 \@changed@x@err{#1}%
9231 }%
9232 \fi
9233 \global\expandafter\let
9234 \csname\cf@encoding\string#1\expandafter\endcsname
9235 \csname ?\string#1\endcsname
9236 \fi
9237 \csname\cf@encoding\string#1%
9238 \expandafter\endcsname
9239 \else
9240 \noexpand#1%
9241 \fi
9242 }
9243 \def\@changed@x@err#1{%
9244 \errhelp{Your command will be ignored, type <return> to proceed}%
9245 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9246 \def\DeclareTextCommandDefault#1{%
9247 \DeclareTextCommand#1?%
9248 }
9249 \def\ProvideTextCommandDefault#1{%
9250 \ProvideTextCommand#1?%
9251 }
9252 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd

```



```

9253 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9254 \def\DeclareTextAccent#1#2#3{%
9255   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9256 }
9257 \def\DeclareTextCompositeCommand#1#2#3#4{%
9258   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9259   \edef\reserved@b{\string##1}%
9260   \edef\reserved@c{%
9261     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9262   \ifx\reserved@b\reserved@c
9263     \expandafter\expandafter\expandafter\ifx
9264       \expandafter\@car\reserved@a\relax\relax\@nil
9265       \@text@composite
9266     \else
9267       \edef\reserved@b##1{%
9268         \def\expandafter\noexpand
9269           \csname#2\string#1\endcsname###1{%
9270             \noexpand\@text@composite
9271             \expandafter\noexpand\csname#2\string#1\endcsname
9272             ###1\noexpand\@empty\noexpand\@text@composite
9273             {##1}%
9274           }%
9275         }%
9276       \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9277     \fi
9278     \expandafter\def\csname\expandafter\string\csname
9279       #2\endcsname\string#1-\string#3\endcsname{#4}
9280   \else
9281     \errhelp{Your command will be ignored, type <return> to proceed}%
9282     \errmessage{\string\DeclareTextCompositeCommand\space used on
9283       inappropriate command \protect#1}
9284   \fi
9285 }
9286 \def\@text@composite#1#2#3\@text@composite{%
9287   \expandafter\@text@composite@x
9288   \csname\string#1-\string#2\endcsname
9289 }
9290 \def\@text@composite@x#1#2{%
9291   \ifx#1\relax
9292     #2%
9293   \else
9294     #1%
9295   \fi
9296 }
9297 %
9298 \def\@strip@args#1:#2-#3\@strip@args{#2}
9299 \def\DeclareTextComposite#1#2#3#4{%
9300   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9301   \bgroup
9302     \lccode`\@=#4%
9303     \lowercase{%
9304       \egroup
9305       \reserved@a \@%
9306     }%
9307 }
9308 %
9309 \def\UseTextSymbol#1#2{#2}
9310 \def\UseTextAccent#1#2#3{}
9311 \def\@use@text@encoding#1{}
9312 \def\DeclareTextSymbolDefault#1#2{%
9313   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}{#1}}%
9314 }
9315 \def\DeclareTextAccentDefault#1#2{%

```

```

9316 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9317 }
9318 \def\cf@encoding{OT1}

```

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

```

9319 \DeclareTextAccent{"}{OT1}{127}
9320 \DeclareTextAccent{'}{OT1}{19}
9321 \DeclareTextAccent{^}{OT1}{94}
9322 \DeclareTextAccent{\`}{OT1}{18}
9323 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9324 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9325 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
9326 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9327 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
9328 \DeclareTextSymbol{\i}{OT1}{16}
9329 \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`.

```

9330 \ifx\scriptsize\undefined
9331 \let\scriptsize\sevenrm
9332 \fi

```

And a few more “dummy” definitions.

```

9333 \def\language{english}%
9334 \let\bbl@opt@shorthands@nnil
9335 \def\bbl@ifshorthand#1#2#3{#2}%
9336 \let\bbl@language@opts@empty
9337 \let\bbl@provide@locale@relax
9338 \ifx\babeloptionstrings\undefined
9339 \let\bbl@opt@strings@nnil
9340 \else
9341 \let\bbl@opt@strings\babeloptionstrings
9342 \fi
9343 \def\BabelStringsDefault{generic}
9344 \def\bbl@tempa{normal}
9345 \ifx\babeloptionmath\bbl@tempa
9346 \def\bbl@mathnormal{\noexpand\textormath}
9347 \fi
9348 \def\AfterBabelLanguage#1#2{}
9349 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9350 \let\bbl@afterlang\relax
9351 \def\bbl@opt@safe{BR}
9352 \ifx\@uclclist\undefined\let\@uclclist\empty\fi
9353 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9354 \expandafter\newif\csname ifbbl@single\endcsname
9355 \chardef\bbl@bidimode\z@
9356 <</Emulate LaTeX>>

```

A proxy file:

```

9357 <*\plain>
9358 \input babel.def
9359 </\plain>

```

15. Acknowledgements

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

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

Barbara Beeton has helped in improving the manual.

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

References

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