

Babel

Code

Version 25.4.79675
2025/03/06

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’	61
4.20	Processing keys in ini	65
4.21	French spacing (again)	70
4.22	Handle language system	71
4.23	Numerals	72
4.24	Casing	74
4.25	Getting info	74
4.26	BCP 47 related commands	76
5	Adjusting the Babel behavior	77
5.1	Cross referencing macros	79
5.2	Layout	81
5.3	Marks	82
5.4	Other packages	83
5.4.1	ifthen	83
5.4.2	varioref	84
5.4.3	hhline	84
5.5	Encoding and fonts	85
5.6	Basic bidi support	86
5.7	Local Language Configuration	90
5.8	Language options	90

6	The kernel of Babel	93
7	Error messages	94
8	Loading hyphenation patterns	97
9	luatex + xetex: common stuff	101
10	Hooks for XeTeX and LuaTeX	105
10.1	XeTeX	105
10.2	Support for interchar	106
10.3	Layout	108
10.4	8-bit TeX	110
10.5	LuaTeX	111
10.6	Southeast Asian scripts	117
10.7	CJK line breaking	119
10.8	Arabic justification	121
10.9	Common stuff	125
10.10	Automatic fonts and ids switching	125
10.11	Bidi	132
10.12	Layout	134
10.13	Lua: transforms	144
10.14	Lua: Auto bidi with basic and basic-r	153
11	Data for CJK	165
12	The ‘nil’ language	165
13	Calendars	166
13.1	Islamic	166
13.2	Hebrew	168
13.3	Persian	172
13.4	Coptic and Ethiopic	173
13.5	Buddhist	173
14	Support for Plain T_EX (plain.def)	174
14.1	Not renaming hyphen.tex	174
14.2	Emulating some L ^A T _E X features	175
14.3	General tools	176
14.4	Encoding related macros	179
15	Acknowledgements	182

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=25.4.79675>>
2 <<date=2025/03/06>>
```

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

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

```

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

```

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

```

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

```

\bbl@afterelse

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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


```

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

```

3.1. A few core definitions

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

```

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

```

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

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

```

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

```

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

3.3. base

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

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

```

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

```

3.4. key=value options and other general option

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

380 \ProcessOptions*

```

3.5. Post-process some options

```

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

```

390 \fi

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

3.6. Plain: babel.def (start)

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

First, exit immediately if previously loaded.

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

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

```

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

```

```

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

```

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

```

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

```

4.1. Selecting the language

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

```

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

```

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

```

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

```

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

```

548 \let\xstring\string

```

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

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

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

```

549 \def\bbl@language@stack{}

```

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

\bbl@push@language

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

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

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

```

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

```

```
639 \select@language{#1}}
```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring `TEX` in a certain pre-defined state.

The name of the language is stored in the control sequence `\language`.

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras{language}` command at definition time by expanding the `\csname` primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\{language\}hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\{language\}hyphenmins` will be used.

No text is supposed to be added with switching captions and date, so we remove any spurious spaces with `\bbl@bsphack` and `\bbl@esphack`.

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

```

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

```

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

```

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

741 \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`.

```

742 \expandafter\def\csname otherlanguage*\endcsname{%
743   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
744 \def\bbl@otherlanguage@s[#1]#2{%
745   \def\bbl@selectorname{other*}%
746   \ifnum\bbl@hymapset=\@ccclv\chardef\bbl@hymapset4\relax\fi
747   \def\bbl@select@opts{#1}%
748   \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”.

```

749 \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.

```

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

```

```

777 \endgroup}
778 \providecommand\BabelWrapText[1]{%
779 \def\bbl@tempa{\def\BabelText###1}%
780 \expandafter\bbl@tempa\expandafter{\BabelText{#1}}}

```

\foreign@language This macro does the work for \foreignlanguage and the otherlanguage* environment. First we need to store the name of the language and check that it is a known language. Then it just calls bbl@switch.

```

781 \def\foreign@language#1{%
782 % set name
783 \edef\language#1%
784 \ifbbl@usedategroup
785 \bbl@add\bbl@select@opts{,date,}%
786 \bbl@usedategroupfalse
787 \fi
788 \bbl@fixname\language
789 \let\localename\language
790 % TODO. name@map here?
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=\@cclv
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\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 `other\language*`.

```

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 $\text{\LaTeX 2}_{\epsilon}$. When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by babel.

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

```

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


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

4.3. More on selection

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

```

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

```

```

955     \expandafter\bbbl@tempb
956     \fi}%
957 \expandafter\bbbl@tempb\bbbl@captionslist\today\@empty
958 \def\bbbl@tempa##1{% elt for include list
959     \ifx##1\@empty\else
960         \bbbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
961         \ifin@else
962             \bbbl@tempb##1\@empty
963         \fi
964     \expandafter\bbbl@tempa
965     \fi}%
966 \bbbl@tempa#1\@empty}
967 \def\bbbl@captionslist{%
968 \prefacename\refname\abstractname\bibname\chaptername\appendixname
969 \contentsname\listfigurename\listtablename\indexname\figurename
970 \tablename\partname\enclname\ccname\headtoname\pagename\seename
971 \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.

```

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

```

4.5. Compatibility with language.def

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

```

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

```

```

1007      \def\uselanguage#1{%
1008      \input language.def
1009      \endgroup
1010      \fi
1011      \fi
1012      \chardef\l@english\z@
1013 \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.

```

1014 \def\addto#1#2{%
1015   \ifx#1\@undefined
1016     \def#1{#2}%
1017   \else
1018     \ifx#1\relax
1019       \def#1{#2}%
1020     \else
1021       {\toks@\expandafter{#1#2}%
1022        \xdef#1{\the\toks@}}%
1023     \fi
1024   \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.

```

1025 \bbl@trace{Hooks}
1026 \newcommand\AddBabelHook[3][[]]{%
1027   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1028   \def\bbl@tempa#1,#3=#2,##3\@empty{\def\bbl@tempb{##3}}%
1029   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1030   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1031     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1032     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1033   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1034 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1035 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1036 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1037 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1038   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1039   \def\bbl@elth##1{%
1040     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}%
1041     \bbl@cs{ev@#2@#3}%
1042     \ifx\language\@undefined\else % Test required for Plain (?)
1043       \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1044       \def\bbl@elth##1{%
1045         \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1#3}}%
1046         \bbl@cs{ev@#2@#1}%
1047       \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).

```

1048 \def\bbl@evargs{,% <- don't delete this comma
1049   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1050   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1051   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1052   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%

```

```

1053 beforestart=0,language=2,begindocument=1}
1054 \ifx\NewHook\@undefined\else % Test for Plain (?)
1055 \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1056 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1057 \fi

```

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

```

1058 \providecommand\PassOptionsToLocale[2]{%
1059 \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.

```

1060 \bbl@trace{Macros for setting language files up}
1061 \def\bbl@ldfinit{%
1062 \let\bbl@screset\@empty
1063 \let\BabelStrings\bbl@opt@string
1064 \let\BabelOptions\@empty
1065 \let\BabelLanguages\relax
1066 \ifx\originalTeX\@undefined
1067 \let\originalTeX\@empty
1068 \else
1069 \originalTeX
1070 \fi}
1071 \def\LdfInit#1#2{%
1072 \chardef\atcatcode=\catcode`\@
1073 \catcode`\@=11\relax
1074 \chardef\eqcatcode=\catcode`\=
1075 \catcode`\==12\relax
1076 \expandafter\if\expandafter\@backslashchar
1077 \expandafter\@car\string#2\@nil
1078 \ifx#2\@undefined\else
1079 \ldf@quit{#1}%
1080 \fi
1081 \else
1082 \expandafter\ifx\csname#2\endcsname\relax\else
1083 \ldf@quit{#1}%
1084 \fi
1085 \fi
1086 \bbl@ldfinit}

```

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

```

1087 \def\ldf@quit#1{%
1088 \expandafter\main@language\expandafter{#1}%
1089 \catcode`\@=\atcatcode \let\atcatcode\relax

```

```

1090 \catcode`\==\eqcatcode \let\eqcatcode\relax
1091 \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.

```

1092 \def\bbl@afterldf#1{%^^A TODO. #1 is not used. Remove
1093 \bbl@afterlang
1094 \let\bbl@afterlang\relax
1095 \let\BabelModifiers\relax
1096 \let\bbl@screset\relax}%
1097 \def\ldf@finish#1{%
1098 \loadlocalcfg{#1}%
1099 \bbl@afterldf{#1}%
1100 \expandafter\main@language\expandafter{#1}%
1101 \catcode`\@=\atcatcode \let\atcatcode\relax
1102 \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1103 \@onlypreamble\LdfInit
1104 \@onlypreamble\ldf@quit
1105 \@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.

```

1106 \def\main@language#1{%
1107 \def\bbl@main@language{#1}%
1108 \let\language\name\bbl@main@language
1109 \let\localename\bbl@main@language
1110 \let\mainlocalename\bbl@main@language
1111 \bbl@id@assign
1112 \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.

```

1113 \def\bbl@beforestart{%
1114 \def\@nolanerr##1{%
1115 \bbl@carg\chardef{l@##1}\z@
1116 \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1117 \bbl@usehooks{beforestart}{}%
1118 \global\let\bbl@beforestart\relax}
1119 \AtBeginDocument{%
1120 {\@nameuse\bbl@beforestart}}% Group!
1121 \if@files
1122 \providecommand\babel@aux[2]{}%
1123 \immediate\write\@mainaux{\unexpanded{%
1124 \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1125 \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1126 \fi
1127 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1128 \ifbbl@single % must go after the line above.
1129 \renewcommand\selectlanguage[1]{}%
1130 \renewcommand\foreignlanguage[2]{#2}%
1131 \global\let\babel@aux\@gobbletwo % Also as flag
1132 \fi}

```

```

1133 %
1134 \ifcase\bbl@engine\or
1135   \AtBeginDocument{\pagedir\bodydir} %^^A TODO - a better place
1136 \fi

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

1137 \def\select@language@x#1{%
1138   \ifcase\bbl@select@type
1139     \bbl@ifsamestring\language#1\{\select@language{#1}}%
1140   \else
1141     \select@language{#1}%
1142   \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.

```

1143 \bbl@trace{Shorhands}
1144 \def\bbl@withactive#1#2{%
1145   \begingroup
1146     \lccode`~=#2\relax
1147     \lowercase{\endgroup#1~}}

```

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

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

```

1148 \def\bbl@add@special#1{% 1:a macro like "\", \?, etc.
1149   \bbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1150   \bbl@ifunset{\@sanitize}\{\bbl@add\@sanitize{\@makeother#1}}%
1151   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1152     \begingroup
1153       \catcode`#1\active
1154       \nfss@catcodes
1155       \ifnum\catcode`#1=\active
1156         \endgroup
1157         \bbl@add\nfss@catcodes{\@makeother#1}%
1158       \else
1159         \endgroup
1160       \fi
1161   \fi}

```

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

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

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

```

1162 \def\bbl@active@def#1#2#3#4{%
1163   \@namedef{#3#1}{%
1164     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1165       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1166     \else
1167       \bbl@afterfi\csname#2@sh@#1@\endcsname
1168     \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.

```

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

```

1175 \def\initiate@active@char#1{%
1176   \bbl@ifunset{active@char\string#1}%
1177   {\bbl@withactive
1178     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1179   {}}

```

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).

```

1180 \def\@initiate@active@char#1#2#3{%
1181   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1182   \ifx#1\@undefined
1183     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}}%
1184   \else
1185     \bbl@csarg\let{oridef@#2}#1%
1186     \bbl@csarg\edef{oridef@#2}{%
1187       \let\noexpand#1%
1188       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1189   \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*").

```

1190   \ifx#1#3\relax
1191     \expandafter\let\csname normal@char#2\endcsname#3%
1192   \else
1193     \bbl@info{Making #2 an active character}%
1194     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1195     \@namedef{normal@char#2}{%
1196       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}}%
1197   \else
1198     \@namedef{normal@char#2}{#3}%
1199   \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).

```

1200   \bbl@restoreactive{#2}%
1201   \AtBeginDocument{%

```

```

1202 \catcode`#2\active
1203 \if@filesw
1204 \immediate\write\@mainaux{\catcode`\string#2\active}%
1205 \fi}%
1206 \expandafter\bbbl@add@special\csname#2\endcsname
1207 \catcode`#2\active
1208 \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⟩`).

```

1209 \let\bbbl@tempa\@firstoftwo
1210 \if\string^#2%
1211 \def\bbbl@tempa{\noexpand\textormath}%
1212 \else
1213 \ifx\bbbl@mathnormal\@undefined\else
1214 \let\bbbl@tempa\bbbl@mathnormal
1215 \fi
1216 \fi
1217 \expandafter\edef\csname active@char#2\endcsname{%
1218 \bbbl@tempa
1219 {\noexpand\if@safe@actives
1220 \noexpand\expandafter
1221 \expandafter\noexpand\csname normal@char#2\endcsname
1222 \noexpand\else
1223 \noexpand\expandafter
1224 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1225 \noexpand\fi}%
1226 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1227 \bbbl@csarg\edef{doactive#2}{%
1228 \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

$$\text{\active@prefix}\langle char\rangle\text{\normal@char}\langle char\rangle$$

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

```

1229 \bbbl@csarg\edef{active@#2}{%
1230 \noexpand\active@prefix\noexpand#1%
1231 \expandafter\noexpand\csname active@char#2\endcsname}%
1232 \bbbl@csarg\edef{normal@#2}{%
1233 \noexpand\active@prefix\noexpand#1%
1234 \expandafter\noexpand\csname normal@char#2\endcsname}%
1235 \bbbl@ncarg\let#1\bbbl@normal@#2}%

```

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

```

1236 \bbbl@active@def#2\user@group{user@active}{language@active}%
1237 \bbbl@active@def#2\language@group{language@active}{system@active}%
1238 \bbbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1239 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1240 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1241 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1242 {\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 `\prim@s` as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```
1243 \if\string'#2%
1244   \let\prim@s\bbl@prim@s
1245   \let\active@math@prime#1%
1246 \fi
1247 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}
```

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

```
1248 <<{*More package options}>> ≡
1249 \DeclareOption{math=active}{}
1250 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1251 <</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*.

```
1252 \ifpackagewith{babel}{KeepShorthandsActive}%
1253   {\let\bbl@restoreactive\@gobble}%
1254   {\def\bbl@restoreactive#1{%
1255     \bbl@exp{%
1256       \\\AfterBabelLanguage\\CurrentOption
1257       {\catcode`#1=\the\catcode`#1\relax}%
1258       \\\AtEndOfPackage
1259       {\catcode`#1=\the\catcode`#1\relax}}}%
1260   \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.

```
1261 \def\bbl@sh@select#1#2{%
1262   \expandafter\ifx\csname#1sh@#2@sel\endcsname\relax
1263     \bbl@afterelse\bbl@scndcs
1264   \else
1265     \bbl@afterfi\csname#1sh@#2@sel\endcsname
1266   \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.

```
1267 \begingroup
1268 \bbl@ifunset{ifincsname}%^^A Ugly. Correct? Only Plain?
1269 {\gdef\active@prefix#1{%
1270   \ifx\protect\@typeset@protect
1271   \else
1272     \ifx\protect\@unexpandable@protect
1273       \noexpand#1%
1274     \else
1275       \protect#1%
1276     \fi
1277   \expandafter\@gobble
1278   \fi}}
1279 {\gdef\active@prefix#1{%
1280   \ifincsname
```

```

1281      \string#1%
1282      \expandafter\@gobble
1283    \else
1284      \ifx\protect\@typeset@protect
1285      \else
1286        \ifx\protect\@unexpandable@protect
1287          \noexpand#1%
1288        \else
1289          \protect#1%
1290        \fi
1291      \expandafter\expandafter\expandafter\@gobble
1292    \fi
1293  \fi}}
1294 \endgroup

```

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

```

1295 \newif\if@safe@actives
1296 \@safe@activefalse

```

\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.

```

1297 \def\bbl@restore@actives{\if@safe@actives\@safe@activefalse\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`.

```

1298 \chardef\bbl@activated\z@
1299 \def\bbl@activate#1{%
1300   \chardef\bbl@activated\@ne
1301   \bbl@withactive{\expandafter\let\expandafter}#1%
1302   \csname bbl@active@string#1\endcsname}
1303 \def\bbl@deactivate#1{%
1304   \chardef\bbl@activated\tw@
1305   \bbl@withactive{\expandafter\let\expandafter}#1%
1306   \csname bbl@normal@string#1\endcsname}

```

\bbl@firstcs

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

```

1307 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1308 \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.

```

1309 \def\babel@texpdf#1#2#3#4{%

```

```

1310 \ifx\texorpdfstring\undefined
1311   \textormath{#1}{#3}%
1312 \else
1313   \texorpdfstring{\textormath{#1}{#3}}{#2}%
1314   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1315 \fi}
1316%
1317 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1318 \def\@decl@short#1#2#3\@nil#4{%
1319   \def\bbl@tempa{#3}%
1320   \ifx\bbl@tempa\@empty
1321     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1322     \bbl@ifunset{#1@sh@\string#2@}{}%
1323     {\def\bbl@tempa{#4}%
1324       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1325       \else
1326         \bbl@info
1327           {Redefining #1 shorthand \string#2\\%
1328            in language \CurrentOption}%
1329       \fi}%
1330     \@namedef{#1@sh@\string#2@}{#4}%
1331   \else
1332     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1333     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1334     {\def\bbl@tempa{#4}%
1335       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1336       \else
1337         \bbl@info
1338           {Redefining #1 shorthand \string#2\string#3\\%
1339            in language \CurrentOption}%
1340       \fi}%
1341     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1342   \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.

```

1343 \def\textormath{%
1344   \ifmmode
1345     \expandafter\@secondoftwo
1346   \else
1347     \expandafter\@firstoftwo
1348   \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’.

```

1349 \def\user@group{user}
1350 \def\language@group{english} %^^A I don't like defaults
1351 \def\system@group{system}

```

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

```

1352 \def\useshorthands{%
1353   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1354   \def\bbl@usesh@s#1{%
1355     \bbl@usesh@x
1356     {\AddBabelHook{babel-sh-\string#1}{afterextras}}{\bbl@activate{#1}}}%
1357   {#1}}

```

```

1358 \def\bbl@usesh@x#1#2{%
1359   \bbl@ifshorthand{#2}%
1360   {\def\user@group{user}%
1361     \initiate@active@char{#2}%
1362     #1%
1363     \bbl@activate{#2}}%
1364   {\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.

```

1365 \def\user@language@group{user@\language@group}
1366 \def\bbl@set@user@generic#1#2{%
1367   \bbl@ifunset{user@generic@active#1}%
1368   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1369     \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1370     \expandafter\edef\csname#2@sh@#1@\endcsname{%
1371       \expandafter\noexpand\csname normal@char#1\endcsname}%
1372     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1373       \expandafter\noexpand\csname user@active#1\endcsname}%
1374     \@empty}
1375 \newcommand\defineshorthand[3][user]{%
1376   \edef\bbl@tempa{\zap@space#1 \@empty}%
1377   \bbl@for\bbl@tempb\bbl@tempa{%
1378     \if*\expandafter\@car\bbl@tempb\@nil
1379     \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1380     \@expandtwoargs
1381     \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1382   \fi
1383   \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.

```

1384 \def\languageshorthands#1{%
1385   \bbl@ifsamestring{none}{#1}{}%
1386   \bbl@once{short-\localename-#1}{%
1387     \bbl@info{'\localename' activates '#1' shorthands.\Reported }}%
1388   \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`.

```

1389 \def\aliasshorthand#1#2{%
1390   \bbl@ifshorthand{#2}%
1391   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1392     \ifx\document@notprerr
1393       \@notshorthand{#2}%
1394     \else
1395       \initiate@active@char{#2}%
1396       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1397       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1398       \bbl@activate{#2}%
1399     \fi
1400   \fi}%
1401   {\bbl@error{shorthand-is-off}{#2}{}}}

```

\@notshorthand

```

1402 \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 `\@nnil` at the end to denote the end of the list of characters.

```
1403 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1404 \DeclareRobustCommand*\shorthandoff{%
1405   \ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1406 \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.

```
1407 \def\bbl@switch@sh#1#2{%
1408   \ifx#2\@nnil\else
1409     \bbl@ifunset{bbl@active@\string#2}%
1410     {\bbl@error{not-a-shorthand-b}{\string#2}}}%
1411     {\ifcase#1%   off, on, off*
1412       \catcode`#2\relax
1413       \or
1414       \catcode`#2\active
1415       \bbl@ifunset{bbl@shdef@\string#2}%
1416       {}%
1417       {\bbl@withactive{\expandafter\let\expandafter}#2%
1418        \csname bbl@shdef@\string#2\endcsname
1419        \bbl@csarg\let{shdef@\string#2}\relax}%
1420       \ifcase\bbl@activated\or
1421       \bbl@activate{#2}%
1422       \else
1423       \bbl@deactivate{#2}%
1424       \fi
1425       \or
1426       \bbl@ifunset{bbl@shdef@\string#2}%
1427       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1428       {}%
1429       \csname bbl@oricat@\string#2\endcsname
1430       \csname bbl@oridef@\string#2\endcsname
1431       \fi}%
1432   \bbl@afterfi\bbl@switch@sh#1%
1433 \fi}
```

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

```
1434 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1435 \def\bbl@putsh#1{%
1436   \bbl@ifunset{bbl@active@\string#1}%
1437   {\bbl@putsh@i#1\@empty\@nnil}%
1438   {\csname bbl@active@\string#1\endcsname}}
1439 \def\bbl@putsh@i#1#2\@nnil{%
1440   \csname\language@group @sh@\string#1@%
1441     \ifx\@empty#2\else\string#2@\fi\endcsname}
1442 %
1443 \ifx\bbl@opt@shorthands\@nnil\else
1444   \let\bbl@s@initiate@active@char\initiate@active@char
1445   \def\initiate@active@char#1{%
1446     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1447   \let\bbl@s@switch@sh\bbl@switch@sh
1448   \def\bbl@switch@sh#1#2{%
1449     \ifx#2\@nnil\else
```

```

1450      \bbl@afterfi
1451      \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1452      \fi}
1453      \let\bbl@s@activate\bbl@activate
1454      \def\bbl@activate#1{%
1455        \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1456      \let\bbl@s@deactivate\bbl@deactivate
1457      \def\bbl@deactivate#1{%
1458        \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1459      \fi

```

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

```

1460 \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.

```

1461 \def\bbl@prim@s{%
1462   \prime\futurelet\@let@token\bbl@pr@m@s}
1463 \def\bbl@if@primes#1#2{%
1464   \ifx#1\@let@token
1465     \expandafter\@firstoftwo
1466   \else\ifx#2\@let@token
1467     \bbl@afterelse\expandafter\@firstoftwo
1468   \else
1469     \bbl@afterfi\expandafter\@secondoftwo
1470   \fi\fi}
1471 \begingroup
1472   \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1473   \catcode`\'=12 \catcode`\"=\active \lccode`\"='\ '
1474   \lowercase{%
1475     \gdef\bbl@pr@m@s{%
1476       \bbl@if@primes" '%
1477         \pr@@@s
1478         {\bbl@if@primes*\^pr@@@t\egroup}}}
1479 \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).

```

1480 \initiate@active@char{~}
1481 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1482 \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.

```

1483 \expandafter\def\csname OT1dqpos\endcsname{127}
1484 \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

```

1485 \ifx\f@encoding\undefined
1486   \def\f@encoding{OT1}
1487 \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.

```
1488 \bbl@trace{Language attributes}
1489 \newcommand\languageattribute[2]{%
1490   \def\bbl@tempc{#1}%
1491   \bbl@fixname\bbl@tempc
1492   \bbl@iflanguage\bbl@tempc{%
1493     \bbl@vforeach{#2}{%
```

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

```
1494     \ifx\bbl@known@attrs\undefined
1495       \in@false
1496     \else
1497       \bbl@xin{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1498     \fi
1499     \ifin@
1500       \bbl@warning{%
1501         You have more than once selected the attribute '##1'\%
1502         for language #1. Reported}%
1503     \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.

```
1504       \bbl@exp{%
1505         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1506       \edef\bbl@tempa{\bbl@tempc-##1}%
1507       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1508       {\csname\bbl@tempc @attr##1\endcsname}%
1509       {\@attrerr{\bbl@tempc}{##1}}%
1510     \fi}}
1511 \@onlypreamble\languageattribute
```

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

```
1512 \newcommand*\@attrerr[2]{%
1513   \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}`.

```
1514 \def\bbl@declare@ttribute#1#2#3{%
1515   \bbl@xin{,#2,}{,\BabelModifiers,}%
1516   \ifin@
1517     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1518   \fi
1519   \bbl@add@list\bbl@attributes{#1-#2}%
1520   \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.

```

1521 \def\bbl@ifattributeset#1#2#3#4{%
1522   \ifx\bbl@known@attribs\@undefined
1523     \in@false
1524   \else
1525     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1526   \fi
1527   \ifin@
1528     \bbl@afterelse#3%
1529   \else
1530     \bbl@afterfi#4%
1531   \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.

```

1532 \def\bbl@ifknown@ttrib#1#2{%
1533   \let\bbl@tempa\@secondoftwo
1534   \bbl@loopx\bbl@tempb{#2}{%
1535     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1536   \ifin@
1537     \let\bbl@tempa\@firstoftwo
1538   \else
1539   \fi}%
1540   \bbl@tempa}

```

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

```

1541 \def\bbl@clear@ttribs{%
1542   \ifx\bbl@attributes\@undefined\else
1543     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1544       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1545     \let\bbl@attributes\@undefined
1546   \fi}
1547 \def\bbl@clear@ttrib#1-#2.{%
1548   \expandafter\let\csname#1@attr#2\endcsname\@undefined}
1549 \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.

```

1550 \bbl@trace{Macros for saving definitions}
1551 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1552 \newcount\babel@savecnt
1553 \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.

```

1554 \def\babel@save#1{%
1555   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1556   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1557     \expandafter{\expandafter,\bbl@savedextras,}}%
1558   \expandafter\in@\bbl@tempa
1559   \ifin@%else
1560     \bbl@add\bbl@savedextras{,{#1,}}%
1561     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1562     \toks@{\expandafter{\originalTeX\let#1=}}%
1563     \bbl@exp{%
1564       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1565     \advance\babel@savecnt@one
1566   \fi}
1567 \def\babel@savevariable#1{%
1568   \toks@{\expandafter{\originalTeX #1=}}%
1569   \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`.

```

1570 \def\bbl@redefine#1{%
1571   \edef\bbl@tempa{\bbl@stripslash#1}%
1572   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1573   \expandafter\def\csname\bbl@tempa\endcsname}
1574 \@onlypreamble\bbl@redefine
```

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

```

1575 \def\bbl@redefine@long#1{%
1576   \edef\bbl@tempa{\bbl@stripslash#1}%
1577   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1578   \long\expandafter\def\csname\bbl@tempa\endcsname}
1579 \@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_`.

```

1580 \def\bbl@redefineroobust#1{%
1581   \edef\bbl@tempa{\bbl@stripslash#1}%
1582   \bbl@ifunset{\bbl@tempa\space}%
1583     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1584       \bbl@exp{\def\\#1\\\\protect<\<\bbl@tempa\space>>}}}%
1585     {\bbl@exp{\let<org@\bbl@tempa><\<\bbl@tempa\space>>}}}%
1586     \@namedef{\bbl@tempa\space}}
1587 \@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.

```

1588 \def\bbl@frenchspacing{%
1589   \ifnum\the\sfcode`\.\=@m
1590     \let\bbl@nonfrenchspacing\relax
1591   \else
1592     \frenchspacing
1593     \let\bbl@nonfrenchspacing\nonfrenchspacing
1594   \fi}
1595 \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.

```

1596 \let\bbl@elt\relax
1597 \edef\bbl@fs@chars{%
1598   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1599   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1600   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1601 \def\bbl@pre@fs{%
1602   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1603   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1604 \def\bbl@post@fs{%
1605   \bbl@save@sfcodes
1606   \edef\bbl@tempa{\bbl@cl{frspc}}%
1607   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1608   \if u\bbl@tempa      % do nothing
1609   \else\if n\bbl@tempa % non french
1610     \def\bbl@elt##1##2##3{%
1611       \ifnum\sfcode`##1=##2\relax
1612         \babel@savevariable{\sfcode`##1}%
1613         \sfcode`##1=##3\relax
1614       \fi}%
1615     \bbl@fs@chars
1616   \else\if y\bbl@tempa % french
1617     \def\bbl@elt##1##2##3{%
1618       \ifnum\sfcode`##1=##3\relax
1619         \babel@savevariable{\sfcode`##1}%
1620         \sfcode`##1=##2\relax
1621       \fi}%
1622     \bbl@fs@chars
1623   \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.

```

1624 \bbl@trace{Hyphens}
1625 \@onlypreamble\babelhyphenation
1626 \AtEndOfPackage{%
1627   \newcommand\babelhyphenation[2][\@empty]{%
1628     \ifx\bbl@hyphenation@\relax
1629       \let\bbl@hyphenation@\@empty
1630     \fi
1631     \ifx\bbl@hyphlist\@empty\else
1632       \bbl@warning{%
1633         You must not intermingle \string\selectlanguage\space and\\%
1634         \string\babelhyphenation\space or some exceptions will not\\%
1635         be taken into account. Reported}%
1636     \fi

```

```

1637 \ifx\@empty#1%
1638 \protected@edef\bb@hyphenation@\bb@hyphenation\space#2}%
1639 \else
1640 \bb@vforeach{#1}{%
1641 \def\bb@tempa{##1}%
1642 \bb@fixname\bb@tempa
1643 \bb@iflanguage\bb@tempa{%
1644 \bb@csarg\protected@edef\hyphenation@\bb@tempa}{%
1645 \bb@ifunset\bb@hyphenation@\bb@tempa}%
1646 }%
1647 {\csname \bb@hyphenation@\bb@tempa\endcsname\space}%
1648 #2}}}%
1649 \fi}}

```

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

```

1650 \ifx\NewDocumentCommand\@undefined\else
1651 \NewDocumentCommand\babelhyphenmins{sommo}{%
1652 \IfNoValueTF{#2}%
1653 {\protected@edef\bb@hyphenmins@\set@hyphenmins{#3}{#4}}%
1654 \IfValueT{#5}{%
1655 \protected@edef\bb@hyphenatmin@\hyphenationmin=#5\relax}}%
1656 \IfBooleanT{#1}{%
1657 \leftthyphenmin=#3\relax
1658 \rightthyphenmin=#4\relax
1659 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1660 {\edef\bb@tempb{\zap@space#2 \@empty}%
1661 \bb@for\bb@tempa\bb@tempb{%
1662 \@namedef\bb@hyphenmins@\bb@tempa}{\set@hyphenmins{#3}{#4}}%
1663 \IfValueT{#5}{%
1664 \@namedef\bb@hyphenatmin@\bb@tempa}{\hyphenationmin=#5\relax}}}%
1665 \IfBooleanT{#1}{\bb@error{hyphenmins-args}{}}}%
1666 \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.

```

1667 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1668 \def\bb@t@one{Tl}
1669 \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.

```

1670 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1671 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1672 \def\bb@hyphen{%
1673 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
1674 \def\bb@hyphen@i#1#2{%
1675 \lowercase{\bb@ifunset\bb@hy@#1#2\@empty}}%
1676 {\csname \bb@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1677 {\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.

```

1678 \def\bb@usehyphen#1{%
1679 \leavevmode

```

```

1680 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1681 \nobreak\hskip\z@skip}
1682 \def\bbl@usehyphen#1{%
1683 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1684 \def\bbl@hyphenchar{%
1685 \ifnum\hyphenchar\font=\m@ne
1686 \babelnullhyphen
1687 \else
1688 \char\hyphenchar\font
1689 \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.

```

1690 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1691 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1692 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1693 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1694 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1695 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1696 \def\bbl@hy@repeat{%
1697 \bbl@usehyphen{
1698 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1699 \def\bbl@hy@@repeat{%
1700 \bbl@usehyphen{
1701 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1702 \def\bbl@hy@empty{\hskip\z@skip}
1703 \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.

```

1704 \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.

```

1705 \bbl@trace{Multiencoding strings}
1706 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1707 <<More package options>> ≡
1708 \DeclareOption{nocase}{}
1709 <</More package options>>

```

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

```

1710 <<More package options>> ≡
1711 \let\bbl@opt@strings\@nnil % accept strings=value
1712 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1713 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1714 \def\BabelStringsDefault{generic}
1715 <</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.

```

1716 \@onlypreamble\StartBabelCommands
1717 \def\StartBabelCommands{%
1718   \begingroup
1719   \@tempcnta="7F
1720   \def\bbbl@tempa{%
1721     \ifnum\@tempcnta>"FF\else
1722       \catcode\@tempcnta=11
1723       \advance\@tempcnta\@ne
1724       \expandafter\bbbl@tempa
1725     \fi}%
1726   \bbbl@tempa
1727   <@Macros local to BabelCommands@>
1728   \def\bbbl@provstring##1##2{%
1729     \providecommand##1{##2}%
1730     \bbbl@tglobal##1}%
1731   \global\let\bbbl@scafter\@empty
1732   \let\StartBabelCommands\bbbl@startcmds
1733   \ifx\BabelLanguages\relax
1734     \let\BabelLanguages\CurrentOption
1735   \fi
1736   \begingroup
1737   \let\bbbl@screset\@nnil % local flag - disable 1st stopcommands
1738   \StartBabelCommands}
1739 \def\bbbl@startcmds{%
1740   \ifx\bbbl@screset\@nnil\else
1741     \bbbl@usehooks{stopcommands}{}%
1742   \fi
1743   \endgroup
1744   \begingroup
1745   \@ifstar
1746     {\ifx\bbbl@opt@strings\@nnil
1747       \let\bbbl@opt@strings\BabelStringsDefault
1748     \fi
1749     \bbbl@startcmds@i}%
1750   \bbbl@startcmds@i}
1751 \def\bbbl@startcmds@i##1##2{%
1752   \edef\bbbl@L{\zap@space#1 \@empty}%
1753   \edef\bbbl@G{\zap@space#2 \@empty}%
1754   \bbbl@startcmds@ii}
1755 \let\bbbl@startcommands\StartBabelCommands

```

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

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

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

```

1756 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1757   \let\SetString\@gobbletwo
1758   \let\bbbl@stringdef\@gobbletwo
1759   \let\AfterBabelCommands\@gobble
1760   \ifx\@empty#1%
1761     \def\bbbl@sc@label{generic}%
1762     \def\bbbl@encstring##1##2{%
1763       \ProvideTextCommandDefault##1{##2}%
1764       \bbbl@tglobal##1%
1765       \expandafter\bbbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1766 \let\bbl@sctest\in@true
1767 \else
1768 \let\bbl@sc@charset\space % <- zapped below
1769 \let\bbl@sc@fontenc\space % <- " "
1770 \def\bbl@tempa##1=##2\@nil{%
1771 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1772 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1773 \def\bbl@tempa##1 ##2{% space -> comma
1774 ##1%
1775 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1776 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1777 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1778 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1779 \def\bbl@encstring##1##2{%
1780 \bbl@foreach\bbl@sc@fontenc{%
1781 \bbl@ifunset{T@####1}%
1782 {}%
1783 {\ProvideTextCommand##1{####1}{##2}%
1784 \bbl@tglobal##1%
1785 \expandafter
1786 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1787 \def\bbl@sctest{%
1788 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1789 \fi
1790 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1791 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1792 \let\AfterBabelCommands\bbl@aftercmds
1793 \let\SetString\bbl@setstring
1794 \let\bbl@stringdef\bbl@encstring
1795 \else % i.e., strings=value
1796 \bbl@sctest
1797 \ifin@
1798 \let\AfterBabelCommands\bbl@aftercmds
1799 \let\SetString\bbl@setstring
1800 \let\bbl@stringdef\bbl@provstring
1801 \fi\fi\fi
1802 \bbl@scswitch
1803 \ifx\bbl@G\@empty
1804 \def\SetString##1##2{%
1805 \bbl@error{missing-group}{##1}{}}}%
1806 \fi
1807 \ifx\@empty#1%
1808 \bbl@usehooks{defaultcommands}{}%
1809 \else
1810 \@expandtwoargs
1811 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1812 \fi}

```

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

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

```

1813 \def\bbl@forlang#1#2{%
1814 \bbl@for#1\bbl@L{%
1815 \bbl@xin@{,##1,}{,\BabelLanguages,}%
1816 \ifin@#2\relax\fi}}
1817 \def\bbl@scswitch{%
1818 \bbl@forlang\bbl@tempa{%
1819 \ifx\bbl@G\@empty\else

```

```

1820     \ifx\SetString@gobbletwo\else
1821     \edef\bbl@GL{\bbl@G\bbl@tempa}%
1822     \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1823     \ifin@else
1824     \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1825     \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1826     \fi
1827     \fi
1828     \fi}}
1829 \AtEndOfPackage{%
1830   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1831   \let\bbl@scswitch\relax}
1832 \@onlypreamble\EndBabelCommands
1833 \def\EndBabelCommands{%
1834   \bbl@usehooks{stopcommands}{}}%
1835   \endgroup
1836   \endgroup
1837   \bbl@scafter}
1838 \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.

```

1839 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1840   \bbl@forlang\bbl@tempa{%
1841     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1842     \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1843     {\bbl@exp{%
1844       \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1845     }%
1846     \def\BabelString{#2}%
1847     \bbl@usehooks{stringprocess}{}}%
1848     \expandafter\bbl@stringdef
1849     \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.

```

1850 \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.

```

1851 << *Macros local to BabelCommands >> ≡
1852 \def\SetStringLoop##1##2{%
1853   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1854   \count@\z@
1855   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1856     \advance\count@\@ne
1857     \toks@\expandafter{\bbl@tempa}%
1858     \bbl@exp{%
1859       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1860       \count@=\the\count@\relax}}}%
1861 << /Macros local to BabelCommands >>

```

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

```

1862 \def\bbl@aftercmds#1{%
1863   \toks@\expandafter{\bbl@scafter#1}%
1864   \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.

```

1865 <<*Macros local to BabelCommands>> ≡
1866   \newcommand\SetCase[3][]{%
1867     \def\bbl@tempa####1####2{%
1868       \ifx####1\@empty\else
1869         \bbl@carg\bbl@add{extras\CurrentOption}{%
1870           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1871           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1872           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1873           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1874         \expandafter\bbl@tempa
1875       \fi}%
1876   \bbl@tempa##1\@empty\@empty
1877   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1878 <</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.

```

1879 <<*Macros local to BabelCommands>> ≡
1880   \newcommand\SetHyphenMap[1]{%
1881     \bbl@forlang\bbl@tempa{%
1882       \expandafter\bbl@stringdef
1883       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1884 <</Macros local to BabelCommands>>

```

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

```

1885 \newcommand\BabelLower[2]{% one to one.
1886   \ifnum\lccode#1=#2\else
1887     \babel@savevariable{\lccode#1}%
1888     \lccode#1=#2\relax
1889   \fi}
1890 \newcommand\BabelLowerMM[4]{% many-to-many
1891   \@tempcnta=#1\relax
1892   \@tempcntb=#4\relax
1893   \def\bbl@tempa{%
1894     \ifnum\@tempcnta>#2\else
1895       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1896       \advance\@tempcnta#3\relax
1897       \advance\@tempcntb#3\relax
1898       \expandafter\bbl@tempa
1899     \fi}%
1900   \bbl@tempa}
1901 \newcommand\BabelLowerM0[4]{% many-to-one
1902   \@tempcnta=#1\relax
1903   \def\bbl@tempa{%
1904     \ifnum\@tempcnta>#2\else
1905       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1906       \advance\@tempcnta#3
1907       \expandafter\bbl@tempa
1908     \fi}%
1909   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1910 <<*More package options>> ≡
1911 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1912 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1913 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1914 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1915 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1916 <</More package options>>

```


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

```

1917 \AtEndOfPackage{%
1918   \ifx\bbbl@opt@hyphenmap\@undefined
1919     \bbbl@xin@{,}\bbbl@language@opts}%
1920     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1921   \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.

```

1922 \newcommand\setlocalecaption{%^A Catch typos.
1923   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1924 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1925   \bbbl@trim@def\bbbl@tempa{#2}%
1926   \bbbl@xin@{.template}\bbbl@tempa}%
1927   \ifin@
1928     \bbbl@ini@captions@template{#3}{#1}%
1929   \else
1930     \edef\bbbl@tempd{%
1931       \expandafter\expandafter\expandafter
1932       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1933     \bbbl@xin@
1934       {\expandafter\string\csname #2name\endcsname}%
1935       {\bbbl@tempd}%
1936     \ifin@ % Renew caption
1937       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
1938     \ifin@
1939       \bbbl@exp{%
1940         \\bbbl@ifsamestring{\bbbl@tempa}{\language name}%
1941         {\\bbbl@scset\<#2name>\<#1#2name>}%
1942         {}}%
1943       \else % Old way converts to new way
1944         \bbbl@ifunset{#1#2name}%
1945         {\bbbl@exp{%
1946           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1947           \\bbbl@ifsamestring{\bbbl@tempa}{\language name}%
1948           {\def\<#2name>\<#1#2name>}}%
1949           {}}}%
1950       {}%
1951     \fi
1952   \else
1953     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
1954     \ifin@ % New way
1955       \bbbl@exp{%
1956         \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
1957         \\bbbl@ifsamestring{\bbbl@tempa}{\language name}%
1958         {\bbbl@scset\<#2name>\<#1#2name>}%
1959         {}}%
1960       \else % Old way, but defined in the new way
1961         \bbbl@exp{%
1962           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1963           \\bbbl@ifsamestring{\bbbl@tempa}{\language name}%
1964           {\def\<#2name>\<#1#2name>}}%
1965           {}}%
1966       \fi%
1967     \fi
1968     \@namedef{#1#2name}{#3}%
1969     \toks@ \expandafter\bbbl@captionslist}%
1970     \bbbl@exp{\in@{\<#2name>}\the\toks@}%
1971     \ifin@ \else
1972       \bbbl@exp{\bbbl@add\bbbl@captionslist{\<#2name>}}%

```

```

1973      \bbl@toglobal\bbl@captionslist
1974      \fi
1975      \fi}
1976 %^^A \def\bbl@setcaption@s#1#2#3{} % Not yet implemented (w/o 'name')

```

4.15. Making glyphs available

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

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

```

1977 \bbl@trace{Macros related to glyphs}
1978 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1979      \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1980      \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.

```

1981 \def\save@sf@q#1{\leavevmode
1982      \begingroup
1983      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1984      \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.

```

1985 \ProvideTextCommand{\quotedblbase}{OT1}{%
1986      \save@sf@q{\set@low@box{\textquotedblright\}}%
1987      \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.

```

1988 \ProvideTextCommandDefault{\quotedblbase}{%
1989      \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

1990 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1991      \save@sf@q{\set@low@box{\textquoteright\}}%
1992      \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.

```

1993 \ProvideTextCommandDefault{\quotesinglbase}{%
1994      \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

1995 \ProvideTextCommand{\guillemetleft}{OT1}{%
1996      \ifmmode
1997          \ll
1998      \else
1999          \save@sf@q{\nobreak
2000              \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2001          \fi}
2002 \ProvideTextCommand{\guillemetright}{OT1}{%
2003      \ifmmode
2004          \gg
2005      \else
2006          \save@sf@q{\nobreak

```

```

2007      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2008 \fi}
2009 \ProvideTextCommand{\guillemotleft}{OT1}{%
2010   \ifmmode
2011     \ll
2012   \else
2013     \save@sf@q{\nobreak
2014       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2015   \fi}
2016 \ProvideTextCommand{\guillemotright}{OT1}{%
2017   \ifmmode
2018     \gg
2019   \else
2020     \save@sf@q{\nobreak
2021       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2022   \fi}

```

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

```

2023 \ProvideTextCommandDefault{\guillemetleft}{%
2024   \UseTextSymbol{OT1}{\guillemetleft}}
2025 \ProvideTextCommandDefault{\guillemetright}{%
2026   \UseTextSymbol{OT1}{\guillemetright}}
2027 \ProvideTextCommandDefault{\guillemotleft}{%
2028   \UseTextSymbol{OT1}{\guillemotleft}}
2029 \ProvideTextCommandDefault{\guillemotright}{%
2030   \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2031 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2032   \ifmmode
2033     <%
2034   \else
2035     \save@sf@q{\nobreak
2036       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2037   \fi}
2038 \ProvideTextCommand{\guilsinglright}{OT1}{%
2039   \ifmmode
2040     >%
2041   \else
2042     \save@sf@q{\nobreak
2043       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2044   \fi}

```

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

```

2045 \ProvideTextCommandDefault{\guilsinglleft}{%
2046   \UseTextSymbol{OT1}{\guilsinglleft}}
2047 \ProvideTextCommandDefault{\guilsinglright}{%
2048   \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.

```

2049 \DeclareTextCommand{\ij}{OT1}{%
2050   i\kern-0.02em\bbl@allowhyphens j}
2051 \DeclareTextCommand{\IJ}{OT1}{%
2052   I\kern-0.02em\bbl@allowhyphens J}
2053 \DeclareTextCommand{\ij}{T1}{\char188}
2054 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2055 \ProvideTextCommandDefault{\ij}{%
2056   \UseTextSymbol{OT1}{\ij}}
2057 \ProvideTextCommandDefault{\IJ}{%
2058   \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).

```
2059 \def\crrtic@{\hrule height0.1ex width0.3em}
2060 \def\crttic@{\hrule height0.1ex width0.33em}
2061 \def\ddj@{%
2062   \setbox0\hbox{d}\dimen@=\ht0
2063   \advance\dimen@lex
2064   \dimen@.45\dimen@
2065   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2066   \advance\dimen@ii.5ex
2067   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2068 \def\DDJ@{%
2069   \setbox0\hbox{D}\dimen@=.55\ht0
2070   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2071   \advance\dimen@ii.15ex % correction for the dash position
2072   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2073   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2074   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2075 %
2076 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2077 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2078 \ProvideTextCommandDefault{\dj}{%
2079   \UseTextSymbol{OT1}{\dj}}
2080 \ProvideTextCommandDefault{\DJ}{%
2081   \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.

```
2082 \DeclareTextCommand{\SS}{OT1}{SS}
2083 \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.

```
2084 \ProvideTextCommandDefault{\glq}{%
2085   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2086 \ProvideTextCommand{\grq}{T1}{%
2087   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2088 \ProvideTextCommand{\grq}{TU}{%
2089   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2090 \ProvideTextCommand{\grq}{OT1}{%
2091   \save@sf@q{\kern-.0125em
2092     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%

```

```

2093 \kern.07em\relax}}
2094 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2095 \ProvideTextCommandDefault{\glqq}{%
2096 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2097 \ProvideTextCommand{\grqq}{T1}{%
2098 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2099 \ProvideTextCommand{\grqq}{TU}{%
2100 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2101 \ProvideTextCommand{\grqq}{0T1}{%
2102 \save@sf@q{\kern-.07em
2103 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2104 \kern.07em\relax}}
2105 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2106 \ProvideTextCommandDefault{\flq}{%
2107 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2108 \ProvideTextCommandDefault{\frq}{%
2109 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```

2110 \ProvideTextCommandDefault{\flqq}{%
2111 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2112 \ProvideTextCommandDefault{\frqq}{%
2113 \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).

```

2114 \def\umlauthigh{%
2115 \def\bbl@umlauta##1{\leavevmode\bgroup%
2116 \accent\csname\f@encoding dqpos\endcsname
2117 ##1\bbl@allowhyphens\egroup}%
2118 \let\bbl@umlaute\bbl@umlauta}
2119 \def\umlautlow{%
2120 \def\bbl@umlauta{\protect\lower@umlaut}}
2121 \def\umlautelow{%
2122 \def\bbl@umlaute{\protect\lower@umlaut}}
2123 \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.

```
2124 \expandafter\ifx\csname U@D\endcsname\relax
2125   \csname newdimen\endcsname\U@D
2126 \fi
```

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

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

```
2127 \def\lower@umlaut#1{%
2128   \leavevmode\bgroup
2129     \U@D lex%
2130     {\setbox\z@\hbox{%
2131       \char\csname f@encoding dqpos\endcsname}%
2132       \dimen@ -.45ex\advance\dimen@ \ht\z@
2133       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2134     \accent\csname f@encoding dqpos\endcsname
2135     \fontdimen5\font\U@D #1%
2136   \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).

```
2137 \AtBeginDocument{%
2138   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2139   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2140   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2141   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
2142   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2143   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2144   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2145   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2146   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2147   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2148   \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.

```
2149 \ifx\l@english\@undefined
2150   \chardef\l@english\z@
2151 \fi
2152 % The following is used to cancel rules in ini files (see Amharic).
2153 \ifx\l@unhyphenated\@undefined
2154   \newlanguage\l@unhyphenated
2155 \fi
```

4.16. Layout

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

```
2156 \bbl@trace{Bidi layout}
2157 \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.

```
2158 \bbl@trace{Input engine specific macros}
2159 \ifcase\bbl@engine
2160   \input txtbabel.def
2161 \or
2162   \input luababel.def
2163 \or
2164   \input xebabel.def
2165 \fi
2166 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2167 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2168 \ifx\babelposthyphenation\undefined
2169   \let\babelposthyphenation\babelprehyphenation
2170   \let\babelpatterns\babelprehyphenation
2171   \let\babelcharproperty\babelprehyphenation
2172 \fi
2173 \end{package} | core
```

4.18. Creating and modifying languages

Continue with \LaTeX only.

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

```
2174 \begin{package}
2175 \bbl@trace{Creating languages and reading ini files}
2176 \let\bbl@extend@ini@gobble
2177 \newcommand\babelprovide[2][]{%
2178   \let\bbl@savelangname\languagename
2179   \edef\bbl@savelocaleid{\the\localeid}%
2180   % Set name and locale id
2181   \edef\languagename{#2}%
2182   \bbl@id@assign
2183   % Initialize keys
2184   \bbl@vforeach{captions,date,import,main,script,language,%
2185     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2186     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2187     Alph,labels,labels*,calendar,date,casing,interchar,@import}%
2188     {\bbl@csarg\let{KVP@##1}\@nnil}%
2189   \global\let\bbl@release@transforms@empty
2190   \global\let\bbl@release@casing@empty
2191   \let\bbl@calendars@empty
2192   \global\let\bbl@inidata@empty
2193   \global\let\bbl@extend@ini@gobble
2194   \global\let\bbl@included@inis@empty
2195   \gdef\bbl@key@list{;}%
2196   \bbl@ifunset{bbl@passto@#2}%
2197     {\def\bbl@tempa{#1}}%
2198     {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}}%
2199   \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2200     \in@{/}{##1}% With /, (re)sets a value in the ini
2201     \ifin@
2202       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2203       \bbl@renewinikey##1\@{##2}%
2204     \else
2205       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2206         \bbl@error{unknown-provide-key}{##1}{}{}%
2207       \fi
2208       \bbl@csarg\def{KVP@##1}{##2}%
2209     \fi}%
\end{package}
```

```

2210 \chardef\bbbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2211 \bbbl@ifunset{date#2}\z@{\bbbl@ifunset{bbbl@llevel@#2}\@ne\tw@}%
2212 % == init ==
2213 \ifx\bbbl@screset\@undefined
2214 \bbbl@ldfinit
2215 \fi
2216 % ==
2217 \ifx\bbbl@KVP@@import\@nnil\else \ifx\bbbl@KVP@import\@nnil
2218 \def\bbbl@KVP@import{\@empty}%
2219 \fi\fi
2220 % == date (as option) ==
2221 % \ifx\bbbl@KVP@date\@nnil\else
2222 % \fi
2223 % ==
2224 \let\bbbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2225 \ifcase\bbbl@howloaded
2226 \let\bbbl@lbfkflag\@empty % new
2227 \else
2228 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2229 \let\bbbl@lbfkflag\@empty
2230 \fi
2231 \ifx\bbbl@KVP@import\@nnil\else
2232 \let\bbbl@lbfkflag\@empty
2233 \fi
2234 \fi
2235 % == import, captions ==
2236 \ifx\bbbl@KVP@import\@nnil\else
2237 \bbbl@exp{\bbbl@ifblank{\bbbl@KVP@import}}%
2238 {\ifx\bbbl@initload\relax
2239 \begingroup
2240 \def\BabelBeforeIni##1##2{\gdef\bbbl@KVP@import{##1}\endinput}%
2241 \bbbl@input@texini{##2}%
2242 \endgroup
2243 \else
2244 \xdef\bbbl@KVP@import{\bbbl@initload}%
2245 \fi}%
2246 {}%
2247 \let\bbbl@KVP@date\@empty
2248 \fi
2249 \let\bbbl@KVP@captions@@\bbbl@KVP@captions
2250 \ifx\bbbl@KVP@captions\@nnil
2251 \let\bbbl@KVP@captions\bbbl@KVP@import
2252 \fi
2253 % ==
2254 \ifx\bbbl@KVP@transforms\@nnil\else
2255 \bbbl@replace\bbbl@KVP@transforms{ }{,}%
2256 \fi
2257 % == Load ini ==
2258 \ifcase\bbbl@howloaded
2259 \bbbl@provide@new{#2}%
2260 \else
2261 \bbbl@ifblank{#1}%
2262 {}% With \bbbl@load@basic below
2263 {\bbbl@provide@renew{#2}}%
2264 \fi
2265 % == include == TODO
2266 % \ifx\bbbl@included@inis\@empty\else
2267 % \bbbl@replace\bbbl@included@inis{ }{,}%
2268 % \bbbl@foreach\bbbl@included@inis{%
2269 % \openin\bbbl@readstream=babel-##1.ini
2270 % \bbbl@extend@ini{#2}}%
2271 % \closein\bbbl@readstream
2272 % \fi

```



```

2273 % Post tasks
2274 % -----
2275 % == subsequent calls after the first provide for a locale ==
2276 \ifx\bbbl@inidata\@empty\else
2277   \bbbl@extend@ini{#2}%
2278 \fi
2279 % == ensure captions ==
2280 \ifx\bbbl@KVP@captions\@nnil\else
2281   \bbbl@ifunset{bbbl@extracaps@#2}%
2282     {\bbbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2283     {\bbbl@exp{\\babelensure[exclude=\\today,
2284       include=\[bbbl@extracaps@#2]]{#2}}}%
2285   \bbbl@ifunset{bbbl@ensure@language}%
2286     {\bbbl@exp{%
2287       \\DeclareRobustCommand<bbbl@ensure@language>[1]{%
2288         \\foreignlanguage{language}%
2289         {###1}}}%
2290     }%
2291   \bbbl@exp{%
2292     \\bbbl@tglobal<bbbl@ensure@language>%
2293     \\bbbl@tglobal<bbbl@ensure@language\space>%
2294 \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.

```

2295 \bbbl@load@basic{#2}%
2296 % == script, language ==
2297 % Override the values from ini or defines them
2298 \ifx\bbbl@KVP@script\@nnil\else
2299   \bbbl@csarg\edef{sname@#2}{\bbbl@KVP@script}%
2300 \fi
2301 \ifx\bbbl@KVP@language\@nnil\else
2302   \bbbl@csarg\edef{lname@#2}{\bbbl@KVP@language}%
2303 \fi
2304 \ifcase\bbbl@engine\or
2305   \bbbl@ifunset{bbbl@chrng@language}{}%
2306     {\directlua{
2307       Babel.set_chranges_b('\bbbl@cl{sbcpr}', '\bbbl@cl{chrng}') }}%
2308 \fi
2309 % == Line breaking: intraspace, intrapenalty ==
2310 % For CJK, East Asian, Southeast Asian, if interspace in ini
2311 \ifx\bbbl@KVP@intraspace\@nnil\else % We can override the ini or set
2312   \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2313 \fi
2314 \bbbl@provide@intraspace
2315 % == Line breaking: justification ==
2316 \ifx\bbbl@KVP@justification\@nnil\else
2317   \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2318 \fi
2319 \ifx\bbbl@KVP@linebreaking\@nnil\else
2320   \bbbl@xin@{,\bbbl@KVP@linebreaking,}%
2321     {,elongated,kashida,cjk,padding,unhyphenated,}%
2322 \ifin@
2323   \bbbl@csarg\xdef
2324     {lnbrk@language}{\expandafter\@car\bbbl@KVP@linebreaking\@nil}%
2325 \fi
2326 \fi
2327 \bbbl@xin@{/e}{/\bbbl@cl{lnbrk}}%
2328 \ifin@ \else \bbbl@xin@{/k}{/\bbbl@cl{lnbrk}} \fi
2329 \ifin@ \bbbl@arabicjust \fi
2330 % WIP
2331 \bbbl@xin@{/p}{/\bbbl@cl{lnbrk}}%

```

```

2332 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2333 % == Line breaking: hyphenate.other.(locale|script) ==
2334 \ifx\bbl@lbfkflag\empty
2335   \bbl@ifunset{bbl@hyotl@language}{%
2336     {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
2337       \bbl@startcommands*{language}{%
2338         \bbl@csarg\bbl@foreach{hyotl@language}{%
2339           \ifcase\bbl@engine
2340             \ifnum##1<257
2341               \SetHyphenMap{\BabelLower{##1}{##1}}%
2342             \fi
2343           \else
2344             \SetHyphenMap{\BabelLower{##1}{##1}}%
2345           \fi}%
2346         \bbl@endcommands}%
2347     \bbl@ifunset{bbl@hyots@language}{%
2348       {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2349         \bbl@csarg\bbl@foreach{hyots@language}{%
2350           \ifcase\bbl@engine
2351             \ifnum##1<257
2352               \global\lccode##1=##1\relax
2353             \fi
2354           \else
2355             \global\lccode##1=##1\relax
2356           \fi}}%
2357   \fi
2358 % == Counters: maparabic ==
2359 % Native digits, if provided in ini (TeX level, xe and lua)
2360 \ifcase\bbl@engine\else
2361   \bbl@ifunset{bbl@dgnat@language}{%
2362     {\expandafter\ifx\csname bbl@dgnat@language\endcsname\empty\else
2363       \expandafter\expandafter\expandafter
2364       \bbl@setdigits\csname bbl@dgnat@language\endcsname
2365       \ifx\bbl@KVP@maparabic@nnil\else
2366         \ifx\bbl@latinarabic@undefined
2367           \expandafter\let\expandafter\@arabic
2368           \csname bbl@counter@language\endcsname
2369         \else % i.e., if layout=counters, which redefines \@arabic
2370           \expandafter\let\expandafter\bbl@latinarabic
2371           \csname bbl@counter@language\endcsname
2372         \fi
2373       \fi
2374     \fi}%
2375 \fi
2376 % == Counters: mapdigits ==
2377 % > luababel.def
2378 % == Counters: alph, Alph ==
2379 \ifx\bbl@KVP@alph@nnil\else
2380   \bbl@exp{%
2381     \\bbl@add\<bbl@preextras@language>{%
2382       \\babel@save\\@alph
2383       \let\\@alph\<bbl@cntr@bbl@KVP@alph @language>}}%
2384 \fi
2385 \ifx\bbl@KVP@Alph@nnil\else
2386   \bbl@exp{%
2387     \\bbl@add\<bbl@preextras@language>{%
2388       \\babel@save\\@Alph
2389       \let\\@Alph\<bbl@cntr@bbl@KVP@Alph @language>}}%
2390 \fi
2391 % == Casing ==
2392 \bbl@release@casing
2393 \ifx\bbl@KVP@casing@nnil\else
2394   \bbl@csarg\xdef{casing@language}%

```

```

2395     {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2396 \fi
2397 % == Calendars ==
2398 \ifx\bbl@KVP@calendar\@nnil
2399   \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2400 \fi
2401 \def\bbl@tempe##1 ##2\@{% % Get first calendar
2402   \def\bbl@tempa{##1}%
2403   \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@}%
2404 \def\bbl@tempe##1.##2.##3\@{%
2405   \def\bbl@tempc{##1}%
2406   \def\bbl@tempb{##2}}%
2407 \expandafter\bbl@tempe\bbl@tempa.\@
2408 \bbl@csarg\edef{calpr@\languagename}{%
2409   \ifx\bbl@tempc@empty\else
2410     calendar=\bbl@tempc
2411   \fi
2412   \ifx\bbl@tempb@empty\else
2413     ,variant=\bbl@tempb
2414   \fi}%
2415 % == engine specific extensions ==
2416 % Defined in XXXbabel.def
2417 \bbl@provide@extra{#2}%
2418 % == require.babel in ini ==
2419 % To load or reload the babel-*.tex, if require.babel in ini
2420 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2421   \bbl@ifunset{bbl@rqtex@\languagename}{}%
2422   {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2423     \let\BabelBeforeIni@gobbletwo
2424     \chardef\atcatcode=\catcode\@
2425     \catcode\@=11\relax
2426     \def\CurrentOption{#2}%
2427     \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2428     \catcode\@=\atcatcode
2429     \let\atcatcode\relax
2430     \global\bbl@csarg\let{rqtex@\languagename}\relax
2431   \fi}%
2432 \bbl@foreach\bbl@calendars{%
2433   \bbl@ifunset{bbl@ca##1}{%
2434     \chardef\atcatcode=\catcode\@
2435     \catcode\@=11\relax
2436     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2437     \catcode\@=\atcatcode
2438     \let\atcatcode\relax}%
2439   {}}%
2440 \fi
2441 % == frenchspacing ==
2442 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2443 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2444 \ifin@
2445   \bbl@extras@wrap{\bbl@pre@fs}%
2446   {\bbl@pre@fs}%
2447   {\bbl@post@fs}%
2448 \fi
2449 % == transforms ==
2450 % > luababel.def
2451 \def\CurrentOption{#2}%
2452 \@nameuse{bbl@icsave#2}%
2453 % == main ==
2454 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2455   \let\languagename\bbl@savelangname
2456   \chardef\localeid\bbl@savelocaleid\relax
2457 \fi

```

```

2458 % == hyphenrules (apply if current) ==
2459 \ifx\bbl@KVP@hyphenrules\@nnil\else
2460   \ifnum\bbl@savelocaleid=\localeid
2461     \language\@nameuse{l@languagename}%
2462   \fi
2463 \fi}

```

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

```

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

```

```

2518 % == hyphenrules (also in new) ==
2519 \ifx\bbbl@lbfkflag\empty
2520   \bbbl@provide@hyphens{#1}%
2521 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values.

```

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

```

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.

```

2544 \def\bbbl@provide@hyphens#1{%
2545   \@tempcnta\m@ne % a flag
2546   \ifx\bbbl@KVP@hyphenrules\@nnil\else
2547     \bbbl@replace\bbbl@KVP@hyphenrules{ }{,}%
2548     \bbbl@foreach\bbbl@KVP@hyphenrules{%
2549       \ifnum\@tempcnta=\m@ne % if not yet found
2550         \bbbl@ifsamestring{##1}{+}%
2551         {\bbbl@carg\addlanguage{l@##1}}%
2552       }%
2553       \bbbl@ifunset{l@##1}% After a possible +
2554       {}%
2555       {\@tempcnta\@nameuse{l@##1}}%
2556     \fi}%
2557   \ifnum\@tempcnta=\m@ne
2558     \bbbl@warning{%
2559       Requested 'hyphenrules' for '\language' not found:\\%
2560       \bbbl@KVP@hyphenrules.\\%
2561       Using the default value. Reported}%
2562   \fi
2563   \fi
2564   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2565     \ifx\bbbl@KVP@captions\@nnil % TODO. Hackish. See above.
2566       \bbbl@ifunset{bbbl@hyphr@#1}{% use value in ini, if exists
2567         {\bbbl@exp{\bbbl@ifblank{\bbbl@cs{hyphr@#1}}}%
2568         }%
2569         {\bbbl@ifunset{l@bbbl@cl{hyphr}}}%
2570         {}% if hyphenrules found:
2571         {\@tempcnta\@nameuse{l@bbbl@cl{hyphr}}}}}%
2572   \fi
2573   \fi
2574   \bbbl@ifunset{l@#1}%

```

```

2575 {\ifnum\@tempcnta=\m@ne
2576 \bbl@carg\adddialect{l@#1}\language
2577 \else
2578 \bbl@carg\adddialect{l@#1}\@tempcnta
2579 \fi}%
2580 {\ifnum\@tempcnta=\m@ne\else
2581 \global\bbl@carg\chardef{l@#1}\@tempcnta
2582 \fi}}

```

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

```

2583 \def\bbl@input@texini#1{%
2584 \bbl@bsphack
2585 \bbl@exp{%
2586 \catcode`\\=14 \catcode`\\=0
2587 \catcode`\\={1 \catcode`\\}=2
2588 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2589 \catcode`\\=\the\catcode`\relax
2590 \catcode`\\=\the\catcode`\relax
2591 \catcode`\\={\the\catcode`\relax
2592 \catcode`\\=\the\catcode`\relax}%
2593 \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.

```

2594 \def\bbl@iniline#1\bbl@iniline{%
2595 \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2596 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2597 \def\bbl@iniskip#1\@@{% if starts with ;
2598 \def\bbl@inistore#1=#2\@@{% full (default)
2599 \bbl@trim@def\bbl@tempa{#1}%
2600 \bbl@trim\toks{#2}%
2601 \bbl@xin@{\bbl@section/\bbl@tempa}{\bbl@key@list}%
2602 \ifin@ \else
2603 \bbl@xin@{,identification/include.}%
2604 {,\bbl@section/\bbl@tempa}%
2605 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2606 \bbl@exp{%
2607 \\\g@addto@macro\\\bbl@inidata{%
2608 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2609 \fi}
2610 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2611 \bbl@trim@def\bbl@tempa{#1}%
2612 \bbl@trim\toks{#2}%
2613 \bbl@xin@{.identification.}{.\bbl@section.}%
2614 \ifin@
2615 \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2616 \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2617 \fi}

```

4.19. Main loop in 'provide'

Now, the 'main loop', which ****must be executed inside a group****. At this point, \bbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

```

2618 \def\bbl@loop@ini{%
2619 \loop
2620 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2621 \endlinechar\m@ne

```

```

2622 \read\bbl@readstream to \bbl@line
2623 \endlinechar\^^M
2624 \ifx\bbl@line\empty\else
2625 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2626 \fi
2627 \repeat}
2628 \ifx\bbl@readstream\undefined
2629 \csname newread\endcsname\bbl@readstream
2630 \fi
2631 \def\bbl@read@ini#1#2{%
2632 \global\let\bbl@extend@ini@gobble
2633 \openin\bbl@readstream=babel-#1.ini
2634 \ifeof\bbl@readstream
2635 \bbl@error{no-ini-file}{#1}{}}%
2636 \else
2637 % == Store ini data in \bbl@inidata ==
2638 \catcode\=[12 \catcode\]=12 \catcode\==12 \catcode\&=12
2639 \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2640 \bbl@info{Importing
2641 \ifcase#2font and identification \or basic \fi
2642 data for \language\name\%
2643 from babel-#1.ini. Reported}%
2644 \ifnum#2=\z@
2645 \global\let\bbl@inidata\empty
2646 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2647 \fi
2648 \def\bbl@section{identification}%
2649 \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
2650 \bbl@inistore load.level=#2\\@@
2651 \bbl@loop@ini
2652 % == Process stored data ==
2653 \bbl@csarg\xdef{lini@language}{#1}%
2654 \bbl@read@ini@aux
2655 % == 'Export' data ==
2656 \bbl@ini@exports{#2}%
2657 \global\bbl@csarg\let{inidata@language}\bbl@inidata
2658 \global\let\bbl@inidata\empty
2659 \bbl@exp{\bbl@add@list\bbl@ini@loaded{language}}%
2660 \bbl@tglobal\bbl@ini@loaded
2661 \fi
2662 \closein\bbl@readstream}
2663 \def\bbl@read@ini@aux{%
2664 \let\bbl@savestrings\empty
2665 \let\bbl@savetoday\empty
2666 \let\bbl@savestate\empty
2667 \def\bbl@elt##1##2##3{%
2668 \def\bbl@section{##1}%
2669 \in@{=date.}{##1}% Find a better place
2670 \ifin@
2671 \bbl@ifunset{bbl@inikv@##1}%
2672 {\bbl@ini@calendar{##1}}%
2673 {}%
2674 \fi
2675 \bbl@ifunset{bbl@inikv@##1}{}%
2676 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2677 \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.

```

2678 \def\bbl@extend@ini@aux#1{%
2679 \bbl@startcommands*{#1}{captions}%
2680 % Activate captions/... and modify exports
2681 \bbl@csarg\def{inikv@captions.licr}##1##2{%

```

```

2682 \setlocalecaption{#1}{##1}{##2}}%
2683 \def\bbl@inikv@captions##1##2{%
2684 \bbl@ini@captions@aux{##1}{##2}}%
2685 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2686 \def\bbl@exportkey##1##2##3{%
2687 \bbl@ifunset{bbl@kv@##2}{%
2688 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2689 \bbl@exp{\global\let\<bbl@##1@\language\>\<bbl@kv@##2>}}%
2690 \fi}}%
2691 % As with \bbl@read@ini, but with some changes
2692 \bbl@read@ini@aux
2693 \bbl@ini@exports\tw@
2694 % Update inidata@lang by pretending the ini is read.
2695 \def\bbl@elt##1##2##3{%
2696 \def\bbl@section{##1}%
2697 \bbl@iniline##2=##3\bbl@iniline}%
2698 \csname bbl@inidata@#1\endcsname
2699 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2700 \StartBabelCommands*{#1}{date}% And from the import stuff
2701 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2702 \bbl@savetoday
2703 \bbl@savestate
2704 \bbl@endcommands}

```

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

```

2705 \def\bbl@ini@calendar#1{%
2706 \lowercase{\def\bbl@tempa{=#1=}}%
2707 \bbl@replace\bbl@tempa{=date.gregorian}{}}%
2708 \bbl@replace\bbl@tempa{=date.}{}}%
2709 \in@{.licr=}{#1=}%
2710 \ifin@
2711 \ifcase\bbl@engine
2712 \bbl@replace\bbl@tempa{.licr=}{}}%
2713 \else
2714 \let\bbl@tempa\relax
2715 \fi
2716 \fi
2717 \ifx\bbl@tempa\relax\else
2718 \bbl@replace\bbl@tempa{=}{}}%
2719 \ifx\bbl@tempa\@empty\else
2720 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2721 \fi
2722 \bbl@exp{%
2723 \def\<bbl@inikv@#1>####1####2{%
2724 \\\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
2725 \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).

```

2726 \def\bbl@renewinikey#1/#2\@#3{%
2727 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2728 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2729 \bbl@trim\toks@{#3}% value
2730 \bbl@exp{%
2731 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2732 \\g@addto@macro\\bbl@inidata{%
2733 \\\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.

```

2734 \def\bbl@exportkey#1#2#3{%
2735 \bbl@ifunset{bbl@kv@#2}%

```



```

2736 {\bbl@csarg\gdef{#1@\\language\name}{#3}}%
2737 {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2738 \bbl@csarg\gdef{#1@\\language\name}{#3}}%
2739 \else
2740 \bbl@exp{\global\let<bbl@#1@\\language\name>\<bbl@kv@#2>}%
2741 \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.

```

2742 \def\bbl@iniwarning#1{%
2743 \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2744 {\bbl@warning{%
2745 From babel-\bbl@cs{lini@\\language\name}.ini:\\%
2746 \bbl@cs{@kv@identification.warning#1}\\%
2747 Reported }}}
2748 %
2749 \let\bbl@release@transforms\@empty
2750 \let\bbl@release@casing\@empty
2751 \def\bbl@ini@exports#1{%
2752 % Identification always exported
2753 \bbl@iniwarning{}}%
2754 \ifcase\bbl@engine
2755 \bbl@iniwarning{.pdf\latex}%
2756 \or
2757 \bbl@iniwarning{.lua\latex}%
2758 \or
2759 \bbl@iniwarning{.xel\latex}%
2760 \fi%
2761 \bbl@exportkey{lllevel}{identification.load.level}{}%
2762 \bbl@exportkey{elname}{identification.name.english}{}%
2763 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2764 {\csname bbl@elname@\\language\name\endcsname}}%
2765 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2766 % Somewhat hackish. TODO:
2767 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2768 \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2769 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2770 \bbl@exportkey{esname}{identification.script.name}{}%
2771 \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
2772 {\csname bbl@esname@\\language\name\endcsname}}%
2773 \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2774 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2775 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2776 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2777 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2778 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2779 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2780 % Also maps bcp47 -> language\name
2781 \ifbbl@bcptoname
2782 \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\\language\name}%
2783 \fi
2784 \ifcase\bbl@engine\or
2785 \directlua{%

```

```

2786     Babel.locale_props[\the\bbl@cs{id@@\languagename}].script
2787     = '\bbl@cl{sbcpr}'}%
2788 \fi
2789 % Conditional
2790 \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
2791   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2792   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2793   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2794   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2795   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2796   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2797   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2798   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2799   \bbl@exportkey{intsp}{typography.intraspace}{}%
2800   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2801   \bbl@exportkey{chrng}{characters.ranges}{}%
2802   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2803   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2804   \ifnum#1=\tw@       % only (re)new
2805     \bbl@exportkey{rqtex}{identification.require.babel}{}%
2806     \bbl@tglobal\bbl@savetoday
2807     \bbl@tglobal\bbl@savestate
2808     \bbl@savestrings
2809   \fi
2810 \fi}

```

4.20. Processing keys in ini

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

```

2811 \def\bbl@inikv#1#2{%      key=value
2812   \toks@{#2}%             This hides #'s from ini values
2813   \bbl@csarg\edef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2814 \let\bbl@inikv@identification\bbl@inikv
2815 \let\bbl@inikv@date\bbl@inikv
2816 \let\bbl@inikv@typography\bbl@inikv
2817 \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.

```

2818 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\languagename}\@empty x-\fi}
2819 \def\bbl@inikv@characters#1#2{%
2820   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2821   {\bbl@exp{%
2822     \\g@addto@macro\\bbl@release@casing{%
2823       \\bbl@casemapping}{\languagename}{\unexpanded{#2}}}%
2824     {\in@{casing.}{#1}% e.g., casing.Uv = uV
2825     \ifin@
2826       \lowercase{\def\bbl@tempb{#1}}%
2827       \bbl@replace\bbl@tempb{casing.}{}%
2828       \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2829         \\bbl@casemapping
2830         {\bbl@maybextx\bbl@tempb}{\languagename}{\unexpanded{#2}}}%
2831       \else
2832         \bbl@inikv{#1}{#2}%
2833     \fi}}

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

2834 \def\bbl@inikv@counters#1#2{%

```

```

2835 \bbl@ifsamestring{#1}{digits}%
2836 {\bbl@error{digits-is-reserved}}{}{}%
2837 {}%
2838 \def\bbl@tempc{#1}%
2839 \bbl@trim@def{\bbl@tempb*}{#2}%
2840 \in@{.1$}{#1$}%
2841 \ifin@
2842 \bbl@replace\bbl@tempc{.1}{}%
2843 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
2844 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2845 \fi
2846 \in@{.F.}{#1}%
2847 \ifin@else\in@{.S.}{#1}\fi
2848 \ifin@
2849 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
2850 \else
2851 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2852 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2853 \bbl@csarg\global\expandafter\let{cntr@#1@\language}\bbl@tempa
2854 \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.

```

2855 \ifcase\bbl@engine
2856 \bbl@csarg\def{inikv@captions.licr}#1#2{%
2857 \bbl@ini@captions@aux{#1}{#2}}
2858 \else
2859 \def\bbl@inikv@captions#1#2{%
2860 \bbl@ini@captions@aux{#1}{#2}}
2861 \fi

```

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

```

2862 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2863 \bbl@replace\bbl@tempa{.template}{}%
2864 \def\bbl@toreplace{#1}{}%
2865 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
2866 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
2867 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2868 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}%
2869 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
2870 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2871 \ifin@
2872 \@nameuse{\bbl@patch\bbl@tempa}%
2873 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2874 \fi
2875 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2876 \ifin@
2877 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2878 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2879 \\\bbl@ifunset{\bbl@bbl@tempa fmt@\\\language}%
2880 {[fnum@\bbl@tempa]}%
2881 {\\\@nameuse{\bbl@bbl@tempa fmt@\\\language}}}}%
2882 \fi}
2883 \def\bbl@ini@captions@aux#1#2{%
2884 \bbl@trim@def\bbl@tempa{#1}%
2885 \bbl@xin@{.template}{\bbl@tempa}%
2886 \ifin@
2887 \bbl@ini@captions@template{#2}\language
2888 \else
2889 \bbl@ifblank{#2}%
2890 {\bbl@exp{%
2891 \toks@{\\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
2892 {\bbl@trim\toks@{#2}}}%

```

```

2893 \bbl@exp{%
2894   \\\bbl@add\\bbl@savestrings{%
2895     \\\SetString<\bbl@tempa name>{\the\toks@}}}%
2896 \toks@%expandafter{\bbl@captionslist}%
2897 \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}}%
2898 \ifin@%else
2899   \bbl@exp{%
2900     \\\bbl@add<\bbl@extracaps@language>{\<\bbl@tempa name>}}%
2901     \\\bbl@tglobal<\bbl@extracaps@language>}}}%
2902 \fi
2903 \fi}

```

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

```

2904 \def\bbl@list@the{%
2905   part,chapter,section,subsection,subsubsection,paragraph,%
2906   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2907   table,page,footnote,mpfootnote,mpfn}
2908 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2909   \bbl@ifunset{\bbl@map@#1@language}%
2910     {\@nameuse{#1}}}%
2911     {\@nameuse{\bbl@map@#1@language}}}%
2912 \def\bbl@inikv@labels#1#2{%
2913   \in@{.map}{#1}%
2914   \ifin@
2915     \ifx\bbl@KVP@labels\@nnil%else
2916       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
2917       \ifin@
2918         \def\bbl@tempc{#1}%
2919         \bbl@replace\bbl@tempc{.map}{}%
2920         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
2921         \bbl@exp{%
2922           \gdef<\bbl@map@bbl@tempc @language>%
2923             {\ifin@<#2>%else\\localecounter{#2}\fi}}}%
2924         \bbl@foreach\bbl@list@the{%
2925           \bbl@ifunset{the##1}{}%
2926           {\bbl@exp{\let\\bbl@tempd<the##1>}}%
2927           \bbl@exp{%
2928             \\\bbl@sreplace<the##1>%
2929             {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2930             \\\bbl@sreplace<the##1>%
2931             {\<\@empty @bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
2932             \expandafter\ifx\csname the##1\endcsname\bbl@tempd%else
2933             \toks@%expandafter\expandafter\expandafter{%
2934               \csname the##1\endcsname}%
2935             \expandafter\xdef\csname the##1\endcsname{\the\toks@}}}%
2936             \fi}}}%
2937 \fi
2938 \fi
2939 %
2940 \else
2941   %
2942   % The following code is still under study. You can test it and make
2943   % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
2944   % language dependent.
2945   \in@{enumerate.}{#1}%
2946   \ifin@
2947     \def\bbl@tempa{#1}%
2948     \bbl@replace\bbl@tempa{enumerate.}{}%
2949     \def\bbl@toreplace{#2}%
2950     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}}%
2951     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2952     \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}}%
2953     \toks@%expandafter{\bbl@toreplace}%

```

```

2954 % TODO. Execute only once:
2955 \bbl@exp{%
2956   \\bbl@add\<extras\language>{%
2957     \\babel@save\<labelenum\romannumeral\bbl@tempa>%
2958     \def\<labelenum\romannumeral\bbl@tempa>\the\toks@}%
2959     \\bbl@tglobal\<extras\language>}%
2960 \fi
2961 \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.

```

2962 \def\bbl@chapttype{chapter}
2963 \ifx\@makechapterhead\undefined
2964 \let\bbl@patchchapter\relax
2965 \else\ifx\thechapter\undefined
2966 \let\bbl@patchchapter\relax
2967 \else\ifx\ps@headings\undefined
2968 \let\bbl@patchchapter\relax
2969 \else
2970 \def\bbl@patchchapter{%
2971   \global\let\bbl@patchchapter\relax
2972   \gdef\bbl@chfmt{%
2973     \bbl@ifunset\bbl@bbl@chapttype fmt@language}%
2974     {\@chapapp\space\thechapter}%
2975     {\@nameuse\bbl@bbl@chapttype fmt@language}}}
2976 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
2977 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
2978 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
2979 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
2980 \bbl@tglobal\appendix
2981 \bbl@tglobal\ps@headings
2982 \bbl@tglobal\chaptermark
2983 \bbl@tglobal\@makechapterhead}
2984 \let\bbl@patchappendix\bbl@patchchapter
2985 \fi\fi\fi
2986 \ifx\@part\undefined
2987 \let\bbl@patchpart\relax
2988 \else
2989 \def\bbl@patchpart{%
2990   \global\let\bbl@patchpart\relax
2991   \gdef\bbl@partformat{%
2992     \bbl@ifunset\bbl@partfmt@language}%
2993     {\partname\nobreakspace\thepart}%
2994     {\@nameuse\bbl@partfmt@language}}}
2995 \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
2996 \bbl@tglobal\@part}
2997 \fi

```

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

```

2998 \let\bbl@calendar\@empty
2999 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3000 \def\bbl@localedate#1#2#3#4{%
3001   \begingroup
3002     \edef\bbl@they{#2}%
3003     \edef\bbl@them{#3}%
3004     \edef\bbl@thed{#4}%
3005     \edef\bbl@tempe{%
3006       \bbl@ifunset\bbl@calpr@language}{\bbl@cl{calpr}},%
3007       #1}%
3008     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3009     \bbl@replace\bbl@tempe{ }{}%

```

```

3010 \bbl@replace\bbl@tempe{convert}{convert=%
3011 \let\bbl@ld@calendar\@empty
3012 \let\bbl@ld@variant\@empty
3013 \let\bbl@ld@convert\relax
3014 \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld###1}{##2}}%
3015 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3016 \bbl@replace\bbl@ld@calendar{gregorian}{}%
3017 \ifx\bbl@ld@calendar\@empty\else
3018 \ifx\bbl@ld@convert\relax\else
3019 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3020 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3021 \fi
3022 \fi
3023 \@nameuse\bbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3024 \edef\bbl@calendar{% Used in \month..., too
3025 \bbl@ld@calendar
3026 \ifx\bbl@ld@variant\@empty\else
3027 .\bbl@ld@variant
3028 \fi}%
3029 \bbl@cased
3030 {\@nameuse\bbl@date@\language @\bbl@calendar}%
3031 \bbl@they\bbl@them\bbl@thed}%
3032 \endgroup}
3033 \def\bbl@printdate#1{%
3034 \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}]
3035 \def\bbl@printdate@i#1[#2]#3#4#5{%
3036 \bbl@usedategroupttrue
3037 \@nameuse\bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}
3038 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3039 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3040 \bbl@trim@def\bbl@tempa{#1.#2}%
3041 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3042 {\bbl@trim@def\bbl@tempa{#3}%
3043 \bbl@trim\toks@{#5}%
3044 \temptokena\expandafter{\bbl@savedate}%
3045 \bbl@exp{% Reverse order - in ini last wins
3046 \def\\bbl@savedate{%
3047 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3048 \the\temptokena}}}%
3049 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3050 {\lowercase{\def\bbl@tempb{#6}}%
3051 \bbl@trim@def\bbl@toreplace{#5}%
3052 \bbl@TG@@date
3053 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3054 \ifx\bbl@savetoday\@empty
3055 \bbl@exp{% TODO. Move to a better place.
3056 \\AfterBabelCommands{%
3057 \gdef\<\language date>{\\protect\<\language date >}%
3058 \gdef\<\language date >{\\bbl@printdate{\language}}}%
3059 \def\\bbl@savetoday{%
3060 \\SetString\\today{%
3061 \<\language date>[convert]%
3062 {\the\year}{\the\month}{\the\day}}}%
3063 \fi}%
3064 {}}}}

```

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).

```

3065 \let\bbl@calendar\@empty
3066 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%

```

```

3067 \nameuse{bbl@ca@#2}#1\@@}
3068 \newcommand\BabelDateSpace{\nobreakspace}
3069 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3070 \newcommand\BabelDated[1]{\number#1}
3071 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3072 \newcommand\BabelDateM[1]{\number#1}
3073 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3074 \newcommand\BabelDateMMM[1]{\fi}
3075 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3076 \newcommand\BabelDatey[1]{\number#1}%
3077 \newcommand\BabelDateyy[1]{\fi}
3078 \ifnum#1<10 0\number#1 %
3079 \else\ifnum#1<100 \number#1 %
3080 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3081 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3082 \else
3083 \bbl@error{limit-two-digits}{\fi}{\fi}%
3084 \fi\fi\fi\fi}
3085 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3086 \newcommand\BabelDateU[1]{\number#1}%
3087 \def\bbl@replace@finish@iii#1{%
3088 \bbl@exp{\def\#1####1####2####3{\the\toks@}}
3089 \def\bbl@TG@@date{%
3090 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3091 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3092 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3093 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3094 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3095 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3096 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3097 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3098 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3099 \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3100 \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3101 \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr{###1|}}%
3102 \bbl@replace\bbl@toreplace{[U]}{\bbl@datecctr{###1|}}%
3103 \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr{###2|}}%
3104 \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr{###3|}}%
3105 \bbl@replace@finish@iii\bbl@toreplace}
3106 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3107 \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.

```

3108 \AddToHook{begindocument/before}{%
3109 \let\bbl@normalsf\normalsfcodes
3110 \let\normalsfcodes\relax}
3111 \AtBeginDocument{%
3112 \ifx\bbl@normalsf\@empty
3113 \ifnum\sfcodes`.=\@m
3114 \let\normalsfcodes\frenchspacing
3115 \else
3116 \let\normalsfcodes\nonfrenchspacing
3117 \fi
3118 \else
3119 \let\normalsfcodes\bbl@normalsf
3120 \fi}

```

Transforms.

```

3121 \bbl@csarg\let\inikv@transforms.prehyphenation\bbl@inikv
3122 \bbl@csarg\let\inikv@transforms.posthyphenation\bbl@inikv

```

```

3123 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3124   #1[#2]{#3}{#4}{#5}}
3125 \begingroup % A hack. TODO. Don't require a specific order
3126   \catcode`\%=12
3127   \catcode`\&=14
3128   \gdef\bbl@transforms#1#2#3{%&
3129     \directlua{
3130       local str = [==[#2]==]
3131       str = str:gsub('%.%d+%.%d+$', '')
3132       token.set_macro('babeltempa', str)
3133     }&%
3134     \def\babeltempc{}&%
3135     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3136     \ifin@else
3137       \bbl@xin@{:,\babeltempa,}{,\bbl@KVP@transforms,}&%
3138     \fi
3139     \ifin@
3140       \bbl@foreach\bbl@KVP@transforms{%&
3141         \bbl@xin@{:,\babeltempa,}{,##1,}&%
3142         \ifin@ &% font:font:transform syntax
3143         \directlua{
3144           local t = {}
3145           for m in string.gmatch('##1'..' ':'', '(.):') do
3146             table.insert(t, m)
3147           end
3148           table.remove(t)
3149           token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))
3150         }&%
3151       \fi}&%
3152     \in@{.0$}{#2$}&%
3153     \ifin@
3154       \directlua{%& (\attribute) syntax
3155         local str = string.match([[ \bbl@KVP@transforms]],
3156           '%([[^%([)]-)%([^%)]-)]-\babeltempa')
3157         if str == nil then
3158           token.set_macro('babeltempb', '')
3159         else
3160           token.set_macro('babeltempb', ',attribute=' .. str)
3161         end
3162       }&%
3163     \toks@{#3}&%
3164     \bbl@exp{%&
3165       \\g@addto@macro\\bbl@release@transforms{%&
3166         \relax &% Closes previous \bbl@transforms@aux
3167         \\bbl@transforms@aux
3168         \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3169         {\language\the\toks@}}&%
3170     \else
3171       \g@addto@macro\bbl@release@transforms{, {#3}}&%
3172     \fi
3173   \fi}
3174 \endgroup

```

4.22. Handle language system

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

```

3175 \def\bbl@provide@lsys#1{%
3176   \bbl@ifunset\bbl@lname@#1{%
3177     {\bbl@load@info{#1}}%
3178   }%
3179   \bbl@csarg\let{lsys@#1}\@empty
3180   \bbl@ifunset\bbl@sname@#1{\bbl@csarg\gdef{sname@#1}{Default}}}%

```



```

3181 \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3182 \bbl@csarg\bbl@add@list{\sys@#1}{Script=\bbl@cs{sname@#1}}%
3183 \bbl@ifunset{\bbl@lname@#1}{}%
3184 {\bbl@csarg\bbl@add@list{\sys@#1}{Language=\bbl@cs{lname@#1}}}%
3185 \ifcase\bbl@engine\or\or
3186 \bbl@ifunset{\bbl@prehc@#1}{}%
3187 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3188 {}%
3189 {\ifx\bbl@xenoxyph\undefined
3190 \global\let\bbl@xenoxyph\bbl@xenoxyph@
3191 \ifx\AtBeginDocument\@notprerr
3192 \expandafter\@secondoftwo % to execute right now
3193 \fi
3194 \AtBeginDocument{%
3195 \bbl@patchfont{\bbl@xenoxyph}%
3196 {\expandafter\select@language\expandafter{\language}}}%
3197 \fi}}%
3198 \fi
3199 \bbl@csarg\bbl@toglobal{\sys@#1}}
3200 \def\bbl@xenoxyph@{%
3201 \bbl@ifset{\bbl@prehc@language}%
3202 {\ifnum\hyphenchar\font=\defaultthyphenchar
3203 \iffontchar\font\bbl@cl{prehc}\relax
3204 \hyphenchar\font\bbl@cl{prehc}\relax
3205 \else\iffontchar\font"200B
3206 \hyphenchar\font"200B
3207 \else
3208 \bbl@warning
3209 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3210 in the current font, and therefore the hyphen\\%
3211 will be printed. Try changing the fontspec's\\%
3212 'HyphenChar' to another value, but be aware\\%
3213 this setting is not safe (see the manual).\\%
3214 Reported}%
3215 \hyphenchar\font\defaultthyphenchar
3216 \fi\fi
3217 \fi}%
3218 {\hyphenchar\font\defaultthyphenchar}}
3219 % \fi}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3220 \def\bbl@load@info#1{%
3221 \def\BabelBeforeIni##1##2{%
3222 \begingroup
3223 \bbl@read@ini{##1}0%
3224 \endinput % babel- .tex may contain onlypreamble's
3225 \endgroup}% boxed, to avoid extra spaces:
3226 {\bbl@input@texini{##1}}}

```

4.23. Numerals

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

```

3227 \def\bbl@setdigits#1#2#3#4#5{%
3228 \bbl@exp{%
3229 \def<\language name digits>###1{% i.e., \langdigits
3230 \<\bbl@digits@language>###1\\\@nil}%
3231 \let\<\bbl@cntr@digits@language>\<\language name digits>%
3232 \def<\language name counter>###1{% i.e., \langcounter

```

[illegible]

```

3258 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3259   \ifx\\#1%           % \\ before, in case #1 is multiletter
3260     \bbl@exp{%
3261       \def\\bbl@tempa####1{%
3262         \<ifcase>####1space\the\toks@\<else>\\<ctrerrr\<fi>}}%
3263     \else
3264       \toks@\expandafter{\the\toks@\or #1}%
3265       \expandafter\bbl@buildifcase
3266     \fi}

```

```

3267 \newcommand\localenumerical[2]{\bbl@cs{cnt@#1@language}{#2}}
3268 \def\bbl@localecnt#1#2{\localenumerical{#2}{#1}}
3269 \newcommand\localecounter[2]{%
3270   \expandafter\bbl@localecnt
3271   \expandafter{\number\csname c@#2\endcsname}{#1}}
3272 \def\bbl@alphnumerical#1#2{%
3273   \expandafter\bbl@alphnumerical@i\number#2 76543210\@@{#1}}
3274 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
3275   \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3276     \bbl@alphnumerical@ii{#9}000000#1\or
3277     \bbl@alphnumerical@ii{#9}00000#1#2\or
3278     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3279     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3280     \bbl@alphnum@invalid{>9999}%
3281   \fi}
3282 \def\bbl@alphnumerical@ii#1#2#3#4#5#6#7#8{%
3283   \bbl@ifunset{bbl@cnt@#1.F.\number#5#6#7#8@language}%
3284     {\bbl@cs{cnt@#1.4@language}{#5}
3285     \bbl@cs{cnt@#1.3@language}{#6}
3286     \bbl@cs{cnt@#1.2@language}{#7}
3287     \bbl@cs{cnt@#1.1@language}{#8}
3288     \ifnum#6#7#8>\z@ % TOD0. An ad hoc rule for Greek. Ugly.

```

```

3289 \bbl@ifunset{bbl@cntr@#1.S.321@\languageName}{}%
3290 {\bbl@cs{cntr@#1.S.321@\languageName}}%
3291 \fi}%
3292 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languageName}}}%
3293 \def\bbl@alphnum@invalid#1{%
3294 \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```

3295 \newcommand\BabelUppercaseMapping[3]{%
3296 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3297 \newcommand\BabelTitlecaseMapping[3]{%
3298 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3299 \newcommand\BabelLowercaseMapping[3]{%
3300 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

The parser for casing and casing.<variant>.
3301 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3302 \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3303 \else
3304 \def\bbl@uftocode#1{\expandafter\string#1}
3305 \fi
3306 \def\bbl@casemapping#1#2#3{% 1:variant
3307 \def\bbl@tempa##1 ##2{% Loop
3308 \bbl@casemapping@i{##1}%
3309 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3310 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3311 \def\bbl@tempe{0}% Mode (upper/lower...)
3312 \def\bbl@tempc{#3}% Casing list
3313 \expandafter\bbl@tempa\bbl@tempc\@empty}
3314 \def\bbl@casemapping@i#1{%
3315 \def\bbl@tempb{#1}%
3316 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3317 \@nameuse{regex_replace_all:nnN}%
3318 {[{\x{c0}-\x{ff}}][{\x{80}-\x{bf}}]*}{\@empty}\bbl@tempb
3319 \else
3320 \@nameuse{regex_replace_all:nnN}{.}{\@empty}\bbl@tempb % TODO. needed?
3321 \fi
3322 \expandafter\bbl@casemapping@ii\bbl@tempb\@empty}
3323 \def\bbl@casemapping@ii#1#2#3\@empty{%
3324 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3325 \ifin@
3326 \edef\bbl@tempe{%
3327 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3328 \else
3329 \ifcase\bbl@tempe\relax
3330 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3331 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3332 \or
3333 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3334 \or
3335 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3336 \or
3337 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3338 \fi
3339 \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.

```

3340 \def\bbl@localeinfo#1#2{%
3341 \bbl@ifunset{bbl@info@#2}{#1}%
3342 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languageName}{#1}%

```

```

3343      {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3344 \newcommand\localeinfo[1]{%
3345   \ifx*#1\@empty    % TODO. A bit hackish to make it expandable.
3346     \bbl@afterelse\bbl@localeinfo{%
3347       \else
3348         \bbl@localeinfo
3349           {\bbl@error{no-ini-info}{}}{}}}%
3350     {#1}%
3351   \fi}
3352 % \@namedef{\bbl@info@name.locale}{lcname}
3353 \@namedef{\bbl@info@tag.ini}{lini}
3354 \@namedef{\bbl@info@name.english}{elname}
3355 \@namedef{\bbl@info@name.opentype}{lname}
3356 \@namedef{\bbl@info@tag.bcp47}{tbcpl}
3357 \@namedef{\bbl@info@language.tag.bcp47}{lbcp}
3358 \@namedef{\bbl@info@tag.opentype}{lotf}
3359 \@namedef{\bbl@info@script.name}{esname}
3360 \@namedef{\bbl@info@script.name.opentype}{sname}
3361 \@namedef{\bbl@info@script.tag.bcp47}{sbcpl}
3362 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3363 \@namedef{\bbl@info@region.tag.bcp47}{rbcp}
3364 \@namedef{\bbl@info@variant.tag.bcp47}{vbcp}
3365 \@namedef{\bbl@info@extension.t.tag.bcp47}{extt}
3366 \@namedef{\bbl@info@extension.u.tag.bcp47}{extu}
3367 \@namedef{\bbl@info@extension.x.tag.bcp47}{extx}

```

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

```

3368 <<*More package options>> ≡
3369 \DeclareOption{ensureinfo=off}{}
3370 <</More package options>>
3371 \let\bbl@ensureinfo\@gobble
3372 \newcommand\BabelEnsureInfo{%
3373   \ifx\InputIfFileExists\@undefined\else
3374     \def\bbl@ensureinfo##1{%
3375       \bbl@ifunset{\bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3376   \fi
3377   \bbl@foreach\bbl@loaded{%
3378     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3379     \def\languagename{##1}%
3380     \bbl@ensureinfo{##1}}}%
3381 \ifpackagewith{babel}{ensureinfo=off}{}%
3382 {\AtEndOfPackage{% Test for plain.
3383   \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3384 \newcommand\getlocaleproperty{%
3385   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3386 \def\bbl@getproperty@s#1#2#3{%
3387   \let#1\relax
3388   \def\bbl@elt##1##2##3{%
3389     \bbl@ifsamestring{##1/##2}{#3}%
3390     {\providecommand#1{##3}%
3391     \def\bbl@elt###1####2####3{}}}%
3392   {}}%
3393   \bbl@cs{inidata@#2}}%
3394 \def\bbl@getproperty@x#1#2#3{%
3395   \bbl@getproperty@s{#1}{#2}{#3}%
3396   \ifx#1\relax
3397     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3398   \fi}
3399 \let\bbl@ini@loaded\@empty
3400 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

```

3401 \def\ShowLocaleProperties#1{%
3402   \typeout{}%
3403   \typeout{*** Properties for language '#1' ***}
3404   \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3405   \@nameuse{bbl@inidata@#1}%
3406   \typeout{*****}}

```

4.26. BCP 47 related commands

```

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

```

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

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

```

3438 \providecommand\BCPdata{}
3439 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3440   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3441   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3442     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3443     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3444     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3445   \def\bbl@bcpdata@ii#1#2{%
3446     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3447     {\bbl@error{unknown-ini-field}{#1}{}}}%
3448     {\bbl@ifunset{bbl\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3449     {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3450 \fi
3451 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3452 \@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.

```

3453 \newcommand\babeladjust[1]{% TODO. Error handling.
3454   \bbl@forkv{#1}{%
3455     \bbl@ifunset{bbl@ADJ@##1@##2}%
3456     {\bbl@cs{ADJ@##1}{##2}}%
3457     {\bbl@cs{ADJ@##1@##2}}}
3458 %
3459 \def\bbl@adjust@lua#1#2{%
3460   \ifvmode
3461     \ifnum\currentgrouplevel=\z@
3462       \directlua{ Babel.#2 }%
3463       \expandafter\expandafter\expandafter\@gobble
3464     \fi
3465   \fi
3466   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3467 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3468   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3469 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3470   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3471 \@namedef{bbl@ADJ@bidi.text@on}{%
3472   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3473 \@namedef{bbl@ADJ@bidi.text@off}{%
3474   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3475 \@namedef{bbl@ADJ@bidi.math@on}{%
3476   \let\bbl@noamsmath\@empty}
3477 \@namedef{bbl@ADJ@bidi.math@off}{%
3478   \let\bbl@noamsmath\relax}
3479 %
3480 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3481   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3482 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3483   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3484 %
3485 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3486   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3487 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3488   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3489 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3490   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3491 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3492   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3493 \@namedef{bbl@ADJ@justify.arabic@on}{%
3494   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3495 \@namedef{bbl@ADJ@justify.arabic@off}{%
3496   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3497 %
3498 \def\bbl@adjust@layout#1{%
3499   \ifvmode
3500     #1%
3501     \expandafter\expandafter\@gobble
3502   \fi
3503   {\bbl@error{layout-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3504 \@namedef{bbl@ADJ@layout.tabular@on}{%
3505   \ifnum\bbl@tabular@mode=\tw@
3506     \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3507   \else
3508     \chardef\bbl@tabular@mode\@ne
3509   \fi}
3510 \@namedef{bbl@ADJ@layout.tabular@off}{%
3511   \ifnum\bbl@tabular@mode=\tw@
3512     \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}%

```

```

3513 \else
3514 \chardef\bbl@tabular@mode\z@
3515 \fi}
3516 \@namedef{bbl@ADJ@layout.lists@on}{%
3517 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3518 \@namedef{bbl@ADJ@layout.lists@off}{%
3519 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3520 %
3521 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3522 \bbl@bcpallowedtrue}
3523 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3524 \bbl@bcpallowedfalse}
3525 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3526 \def\bbl@bcp@prefix{#1}}
3527 \def\bbl@bcp@prefix{bcp47-}
3528 \@namedef{bbl@ADJ@autoload.options#1}{%
3529 \def\bbl@autoload@options{#1}}
3530 \def\bbl@autoload@bcptoptions{import}
3531 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3532 \def\bbl@autoload@bcptoptions{#1}}
3533 \newif\ifbbl@bcptname
3534 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3535 \bbl@bcptnametrue}
3536 \BabelEnsureInfo}
3537 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3538 \bbl@bcptnamefalse}
3539 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3540 \directlua{ Babel.ignore_pre_char = function(node)
3541 return (node.lang == \the\csname l@nohyphenation\endcsname)
3542 end }}
3543 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3544 \directlua{ Babel.ignore_pre_char = function(node)
3545 return false
3546 end }}
3547 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3548 \def\bbl@ignoreinterchar{%
3549 \ifnum\language=\l@nohyphenation
3550 \expandafter\@gobble
3551 \else
3552 \expandafter\@firstofone
3553 \fi}}
3554 \@namedef{bbl@ADJ@interchar.disable@off}{%
3555 \let\bbl@ignoreinterchar\@firstofone}
3556 \@namedef{bbl@ADJ@select.write@shift}{%
3557 \let\bbl@restorelastskip\relax
3558 \def\bbl@savelastskip{%
3559 \let\bbl@restorelastskip\relax
3560 \ifvmode
3561 \ifdim\lastskip=\z@
3562 \let\bbl@restorelastskip\nobreak
3563 \else
3564 \bbl@exp{%
3565 \def\\bbl@restorelastskip{%
3566 \skip@=\the\lastskip
3567 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3568 \fi
3569 \fi}}
3570 \@namedef{bbl@ADJ@select.write@keep}{%
3571 \let\bbl@restorelastskip\relax
3572 \let\bbl@savelastskip\relax}
3573 \@namedef{bbl@ADJ@select.write@omit}{%
3574 \AddBabelHook{babel-select}{beforestart}{%
3575 \expandafter\babel@aux\expandafter{\bbl@main@language}}}%

```

```

3576 \let\bbl@restorelastskip\relax
3577 \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3578 \@namedef{bbl@ADJ@select.encoding@off}{%
3579 \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.

```

3580 << *More package options >> ≡
3581 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3582 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3583 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3584 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3585 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3586 << /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.

```

3587 \bbl@trace{Cross referencing macros}
3588 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3589 \def\@newl@bel#1#2#3{%
3590 {\@safe@activetrue
3591 \bbl@ifunset{#1@#2}%
3592 \relax
3593 {\gdef\@multiplelabels{%
3594 \latex@warning@no@line{There were multiply-defined labels}}%
3595 \latex@warning@no@line{Label `#2' multiply defined}}%
3596 \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.

```

3597 \CheckCommand*\@testdef[3]{%
3598 \def\reserved@a{#3}%
3599 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3600 \else
3601 \@tempswatrue
3602 \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.

```

3603 \def\@testdef#1#2#3{% TODO. With @samestring?
3604 \@safe@activetrue
3605 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3606 \def\bbl@tempb{#3}%
3607 \@safe@activesfalse
3608 \ifx\bbl@tempa\relax
3609 \else
3610 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3611 \fi

```



```

3612 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3613 \ifx\bbl@tempa\bbl@tempb
3614 \else
3615 \@tempswattrue
3616 \fi}
3617 \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.

```

3618 \bbl@xin@{R}\bbl@opt@safe
3619 \ifin@
3620 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3621 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3622 {\expandafter\strip@prefix\meaning\ref}%
3623 \ifin@
3624 \bbl@redefine\@kernel@ref#1{%
3625 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activesfalse}
3626 \bbl@redefine\@kernel@pageref#1{%
3627 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3628 \bbl@redefine\@kernel@sref#1{%
3629 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activesfalse}
3630 \bbl@redefine\@kernel@spageref#1{%
3631 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3632 \else
3633 \bbl@redefineroquest\ref#1{%
3634 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3635 \bbl@redefineroquest\pageref#1{%
3636 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3637 \fi
3638 \else
3639 \let\org@ref\ref
3640 \let\org@pageref\pageref
3641 \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.

```

3642 \bbl@xin@{B}\bbl@opt@safe
3643 \ifin@
3644 \bbl@redefine\@citex[#1]#2{%
3645 \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activesfalse
3646 \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.)

```

3647 \AtBeginDocument{%
3648 \ifpackageloaded{natbib}{%
3649 \def\@citex[#1][#2]#3{%
3650 \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activesfalse
3651 \org@@citex[#1][#2]{\bbl@tempa}}%
3652 }{}}

```

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

```

3653 \AtBeginDocument{%
3654   \ifpackageloaded{cite}{%
3655     \def\citex[#1]#2{%
3656       \@safe@activetrue\org@citex[#1]{#2}\@safe@activesfalse}%
3657     }{}}

```

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

```

3658 \bbl@redefine\nocite#1{%
3659   \@safe@activetrue\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@activetrue is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order to determine during aux file processing which definition of \bibcite is needed we define \bibcite in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

```

3660 \bbl@redefine\bibcite{%
3661   \bbl@cite@choice
3662   \bibcite}

```

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

```

3663 \def\bbl@bibcite#1#2{%
3664   \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.

```

3665 \def\bbl@cite@choice{%
3666   \global\let\bibcite\bbl@bibcite
3667   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3668     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3669     \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.

```

3670 \AtBeginDocument{\bbl@cite@choice}

```

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

```

3671 \bbl@redefine\@bibitem#1{%
3672   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
3673 \else
3674   \let\org@nocite\nocite
3675   \let\org@citex\citex
3676   \let\org@bibcite\bibcite
3677   \let\org@bibitem\@bibitem
3678 \fi

```

5.2. Layout

```

3679 \newcommand\BabelPatchSection[1]{%
3680   \ifundefined{#1}{%
3681     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3682     \@namedef{#1}{%
3683       \@ifstar{\bbl@presec@s{#1}}%

```

```

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

```

5.3. Marks

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

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

```

3712 \bbl@trace{Marks}
3713 \IfBabelLayout{sectioning}
3714   {\ifx\bbl@opt@headfoot\@nnil
3715     \g@addto@macro\@resetactivechars{%
3716       \set@typeset@protect
3717       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3718       \let\protect\noexpand
3719       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3720         \edef\thepage{%
3721           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3722       \fi}%
3723   \fi}
3724 {\ifbbl@single\else
3725   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
3726     \markright#1{%
3727       \bbl@ifblank{#1}%
3728       {\org@markright{}}}%
3729     {\toks@{#1}%
3730       \bbl@exp{%
3731         \\\org@markright{\\protect\\foreignlanguage{\language}%
3732           {\\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 `\mkboth`. (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.)

```

3733 \ifx\mkboth\mkboth
3734 \def\bbl@tempc{\let\mkboth\mkboth}%
3735 \else
3736 \def\bbl@tempc{%
3737 \fi
3738 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefineroobust
3739 \markboth#1#2{%
3740 \protected@edef\bbl@tempb##1{%
3741 \protect\foreignlanguage
3742 {\language}\protect\bbl@restore@actives##1}%
3743 \bbl@ifblank{#1}%
3744 {\toks@}%
3745 {\toks@\expandafter{\bbl@tempb{#1}}}%
3746 \bbl@ifblank{#2}%
3747 {\@temptokena}%
3748 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3749 \bbl@exp{\org@markboth{the\toks@}{the\@temptokena}}}%
3750 \bbl@tempc
3751 \fi} % end ifbbl@single, end \IfBabelLayout

```

5.4. Other packages

5.4.1. `ifthen`

`\ifthenelse` Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

```

% \ifthenelse{isodd{\pageref{some-label}}}
% {code for odd pages}
% {code for even pages}
%

```

In order for this to work the argument of `\isodd` needs to be fully expandable. With the above redefinition of `\pageref` it is not in the case of this example. To overcome that, we add some code to the definition of `\ifthenelse` to make things work.

We want to revert the definition of `\pageref` and `\ref` to their original definition for the first argument of `\ifthenelse`, so we first need to store their current meanings.

Then we can set the `\@safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments.

```

3752 \bbl@trace{Preventing clashes with other packages}
3753 \ifx\org@ref@undefined\else
3754 \bbl@xin@{R}\bbl@opt@safe
3755 \ifin@
3756 \AtBeginDocument{%
3757 \ifpackageloaded{ifthen}{%
3758 \bbl@redefine@long\ifthenelse#1#2#3{%
3759 \let\bbl@temp@pref\pageref
3760 \let\pageref\org@pageref
3761 \let\bbl@temp@ref\ref
3762 \let\ref\org@ref
3763 \@safe@activestrue
3764 \org@ifthenelse{#1}%
3765 {\let\pageref\bbl@temp@pref
3766 \let\ref\bbl@temp@ref
3767 \@safe@activesfalse
3768 #2}%
3769 {\let\pageref\bbl@temp@pref
3770 \let\ref\bbl@temp@ref

```

```

3771         \@safe@activesfalse
3772         #3}%
3773     }%
3774 }{}%
3775 }
3776 \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.

```

3777 \AtBeginDocument{%
3778   \@ifpackageloaded{varioref}{%
3779     \bbl@redefine\@@vpageref#1[#2]#3{%
3780       \@safe@activestrue
3781       \org@@vpageref{#1}[#2]#3}%
3782     \@safe@activesfalse}%
3783   \bbl@redefine\vrefpagenum#1#2{%
3784     \@safe@activestrue
3785     \org@vrefpagenum{#1}#2}%
3786   \@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.

```

3787   \expandafter\def\csname Ref \endcsname#1{%
3788     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3789   }{}%
3790 }
3791 \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.

```

3792 \AtEndOfPackage{%
3793   \AtBeginDocument{%
3794     \@ifpackageloaded{hhline}%
3795     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3796       \else
3797         \makeatletter
3798         \def\@currname{hhline}\input{hhline.sty}\makeatother
3799       \fi}%
3800     {}}}

```

\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 L^AT_EX (\DeclareFontFamilySubstitution).

```

3801 \def\substitutefontfamily#1#2#3{%
3802   \lowercase{\immediate\openout15=#1#2.f\d\relax}%
3803   \immediate\write15{%
3804     \string\ProvidesFile{#1#2.f\d}%
3805     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3806     \space generated font description file]^\j

```

```

3807 \string\DeclareFontFamily{#1}{#2}{}}^^J
3808 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^^J
3809 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^^J
3810 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^^J
3811 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^^J
3812 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^^J
3813 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^^J
3814 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^^J
3815 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^^J
3816 }%
3817 \closeout15
3818 }
3819 \@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

```

3820 \bbl@trace{Encoding and fonts}
3821 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3822 \newcommand\BabelNonText{TS1,T3,TS3}
3823 \let\org@TeX\TeX
3824 \let\org@LaTeX\LaTeX
3825 \let\ensureascii@firstofone
3826 \let\asciienencoding@empty
3827 \AtBeginDocument{%
3828   \def\elt#1{,#1,}%
3829   \edef\bbl@tempa{\expandafter\@gobbletwo\fontenc@load@list}%
3830   \let\elt\relax
3831   \let\bbl@tempb@empty
3832   \def\bbl@tempc{OT1}%
3833   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3834     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3835   \bbl@foreach\bbl@tempa{%
3836     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3837     \ifin@
3838       \def\bbl@tempb{#1}% Store last non-ascii
3839     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3840       \ifin@
3841       \def\bbl@tempc{#1}% Store last ascii
3842     \fi
3843   \fi}%
3844   \ifx\bbl@tempb@empty\else
3845     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3846     \ifin@
3847       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3848     \fi
3849     \let\asciienencoding\bbl@tempc
3850     \renewcommand\ensureascii[1]{%
3851       {\fontencoding{\asciienencoding}\selectfont#1}}%
3852     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3853     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3854   \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.

```
3855 \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.

```
3856 \AtBeginDocument{%
3857   \ifpackageloaded{fontspec}%
3858     {\xdef\latinencoding{%
3859       \ifx\UTFencname\@undefined
3860         EU\ifcase\bbl@engine\or2\or1\fi
3861       \else
3862         \UTFencname
3863       \fi}}%
3864   {\gdef\latinencoding{OT1}%
3865     \ifx\cf@encoding\bbl@t@one
3866       \xdef\latinencoding{\bbl@t@one}%
3867     \else
3868       \def\@elt#1{,#1,}%
3869       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3870       \let\@elt\relax
3871       \bbl@xin@{,T1,}\bbl@tempa
3872       \ifin@
3873         \xdef\latinencoding{\bbl@t@one}%
3874       \fi
3875     \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.

```
3876 \DeclareRobustCommand{\latintext}{%
3877   \fontencoding{\latinencoding}\selectfont
3878   \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.

```
3879 \ifx\@undefined\DeclareTextFontCommand
3880   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3881 \else
3882   \DeclareTextFontCommand{\textlatin}{\latintext}
3883 \fi
```

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

```
3884 \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.

```

3885 \bbl@trace{Loading basic (internal) bidi support}
3886 \ifodd\bbl@engine
3887 \else % TODO. Move to txtbabel. Any xe+lua bidi
3888   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3889     \bbl@error{bidi-only-lua}{}}{}%
3890     \let\bbl@beforeforeign\leavevmode
3891     \AtEndOfPackage{%
3892       \EnableBabelHook{babel-bidi}%
3893       \bbl@xebidipar}
3894 \fi\fi
3895 \def\bbl@loadxebidi#1{%
3896   \ifx\RTLfootnotetext\@undefined
3897     \AtEndOfPackage{%
3898       \EnableBabelHook{babel-bidi}%
3899       \ifx\fontspec\@undefined
3900         \usepackage{fontspec}% bidi needs fontspec
3901       \fi
3902       \usepackage#1{bidi}%
3903       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3904       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3905         \ifnum\@nameuse{\bbl@wdir\@languagename}=\tw@ % 'AL' bidi
3906           \bbl@digitsdotdash % So ignore in 'R' bidi
3907         \fi}}%
3908   \fi}
3909 \ifnum\bbl@bidimode>200 % Any xe bidi=
3910   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3911     \bbl@tentative{bidi=bidi}
3912     \bbl@loadxebidi{}
3913   \or
3914     \bbl@loadxebidi{[rldocument]}
3915   \or
3916     \bbl@loadxebidi{}
3917   \fi
3918 \fi
3919 \fi
3920 % TODO? Separate:
3921 \ifnum\bbl@bidimode=\@ne % bidi=default
3922   \let\bbl@beforeforeign\leavevmode
3923   \ifodd\bbl@engine % lua
3924     \newattribute\bbl@attr@dir
3925     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3926     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3927   \fi
3928   \AtEndOfPackage{%
3929     \EnableBabelHook{babel-bidi}% pdf/lua/xe
3930     \ifodd\bbl@engine\else % pdf/xe
3931       \bbl@xebidipar
3932     \fi}
3933 \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.


```

3934 \bbl@trace{Macros to switch the text direction}
3935 \def\bbl@alscripts{%
3936   ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
3937 \def\bbl@rscripts{%
3938   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
3939   Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
3940   Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaeen,%
3941   Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
3942   Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
3943   Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
3944   Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
3945   Meroitic,N'Ko,Orkhon,Todhri}
3946 \def\bbl@provide@dirs#1{%
3947   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3948   \ifin@
3949     \global\bbl@csarg\chardef{wdir@#1}\@ne
3950     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3951     \ifin@
3952       \global\bbl@csarg\chardef{wdir@#1}\tw@
3953       \fi
3954   \else
3955     \global\bbl@csarg\chardef{wdir@#1}\z@
3956   \fi
3957   \ifodd\bbl@engine
3958     \bbl@csarg\ifcase{wdir@#1}%
3959       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
3960     \or
3961       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
3962     \or
3963       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
3964     \fi
3965   \fi}
3966 \def\bbl@switchdir{%
3967   \bbl@ifunset{\bbl@sys@\languagename}{\bbl@provide@sys{\languagename}}{}%
3968   \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3969   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
3970 \def\bbl@setdirs#1{% TODO - math
3971   \ifcase\bbl@select@type % TODO - strictly, not the right test
3972     \bbl@bodydir{#1}%
3973     \bbl@pardir{#1}% <- Must precede \bbl@texdir
3974   \fi
3975   \bbl@texdir{#1}}
3976 \ifnum\bbl@bidimode>\z@
3977   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3978   \DisableBabelHook{babel-bidi}
3979 \fi

```

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

```

3980 \ifodd\bbl@engine % luatex=1
3981 \else % pdftex=0, xetex=2
3982   \newcount\bbl@dirlevel
3983   \chardef\bbl@thetextdir\z@
3984   \chardef\bbl@thepardir\z@
3985   \def\bbl@texdir#1{%
3986     \ifcase#1\relax
3987       \chardef\bbl@thetextdir\z@
3988       \@nameuse{setlatin}%
3989       \bbl@texdir@i\beginL\endL
3990     \else
3991       \chardef\bbl@thetextdir\@ne
3992       \@nameuse{setnonlatin}%
3993       \bbl@texdir@i\beginR\endR
3994     \fi}

```

```

3995 \def\bbl@textdir@i#1#2{%
3996   \ifhmode
3997     \ifnum\currentgrouplevel>\z@
3998       \ifnum\currentgrouplevel=\bbl@dirlevel
3999         \bbl@error{multiple-bidi}{\}\}%
4000         \bgroup\aftergroup#2\aftergroup\egroup
4001       \else
4002         \ifcase\currentgrouptype\or % 0 bottom
4003           \aftergroup#2% 1 simple {}
4004         \or
4005           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4006         \or
4007           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4008         \or\or\or % vbox vtop align
4009         \or
4010           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4011         \or\or\or\or\or\or % output math disc insert vcent mathchoice
4012         \or
4013           \aftergroup#2% 14 \beginR
4014         \else
4015           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4016         \fi
4017       \fi
4018       \bbl@dirlevel\currentgrouplevel
4019     \fi
4020   #1%
4021 \fi}
4022 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4023 \let\bbl@bodydir\@gobble
4024 \let\bbl@pagedir\@gobble
4025 \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).

```

4026 \def\bbl@xebidipar{%
4027   \let\bbl@xebidipar\relax
4028   \TeXeTstate\@ne
4029   \def\bbl@xeeverypar{%
4030     \ifcase\bbl@thepardir
4031       \ifcase\bbl@thetextdir\else\beginR\fi
4032     \else
4033       {\setbox\z@\lastbox\beginR\box\z@}%
4034     \fi}%
4035   \AddToHook{para/begin}{\bbl@xeeverypar}}
4036 \ifnum\bbl@bidimode>200 % Any xe bidi=
4037   \let\bbl@textdir@i\@gobbletwo
4038   \let\bbl@xebidipar\@empty
4039   \AddBabelHook{bidi}{foreign}{%
4040     \ifcase\bbl@thetextdir
4041       \BabelWrapText{\LR{##1}}%
4042     \else
4043       \BabelWrapText{\RL{##1}}%
4044     \fi}
4045   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4046 \fi
4047 \fi

```

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

```

4048 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}
4049 \AtBeginDocument{%
4050   \ifx\pdfstringdefDisableCommands\undefined\else
4051     \ifx\pdfstringdefDisableCommands\relax\else
4052       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%

```

```

4053 \fi
4054 \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.

```

4055 \bbl@trace{Local Language Configuration}
4056 \ifx\loadlocalcfg\undefined
4057 \ifpackagewith{babel}{noconfigs}%
4058 {\let\loadlocalcfg@gobble}%
4059 {\def\loadlocalcfg#1{%
4060 \InputIfFileExists{#1.cfg}%
4061 {\typeout{*****^J%
4062 * Local config file #1.cfg used^^J%
4063 *}}%
4064 \@empty}}
4065 \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).

```

4066 \bbl@trace{Language options}
4067 \let\bbl@afterlang\relax
4068 \let\BabelModifiers\relax
4069 \let\bbl@loaded\@empty
4070 \def\bbl@load@language#1{%
4071 \InputIfFileExists{#1.ldf}%
4072 {\edef\bbl@loaded{\CurrentOption
4073 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4074 \expandafter\let\expandafter\bbl@afterlang
4075 \csname\CurrentOption.ldf-h@k\endcsname
4076 \expandafter\let\expandafter\BabelModifiers
4077 \csname bbl@mod@\CurrentOption\endcsname
4078 \bbl@exp{\AtBeginDocument{%
4079 \bbl@usehooks@lang{\CurrentOption}{\begin{document}}{\CurrentOption}}}%
4080 {\IfFileExists{babel-#1.tex}%
4081 {\def\bbl@tempa{%
4082 .\There is a locale ini file for this language.\%
4083 If it's the main language, try adding `provide=*'\%
4084 to the babel package options}}%
4085 {\let\bbl@tempa\empty}%
4086 \bbl@error{unknown-package-option}{}}}}

```

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

```

4087 \def\bbl@try@load@lang#1#2#3{%
4088 \IfFileExists{\CurrentOption.ldf}%
4089 {\bbl@load@language{\CurrentOption}}%
4090 {#1\bbl@load@language{#2}#3}}
4091 %
4092 \DeclareOption{friulian}{\bbl@try@load@lang}{friulan}}
4093 \DeclareOption{hebrew}{%
4094 \ifcase\bbl@engine\or
4095 \bbl@error{only-pdftex-lang}{hebrew}{luatex}}%

```

```

4096 \fi
4097 \input{rlbabel.def}%
4098 \bbl@load@language{hebrew}}
4099 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4100 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4101 % \DeclareOption{northernkurdish}{\bbl@try@load@lang{}{kurmanji}{}}
4102 \DeclareOption{polutonikogreek}{%
4103   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4104 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4105 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4106 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

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

```

4107 \ifx\bbl@opt@config\@nnil
4108   \ifpackagewith{babel}{noconfigs}{}%
4109     {\InputIfFileExists{bblopts.cfg}%
4110      {\typeout{*****^J%
4111                * Local config file bblopts.cfg used^^J%
4112                *}}}%
4113     {}}%
4114 \else
4115   \InputIfFileExists{\bbl@opt@config.cfg}%
4116   {\typeout{*****^J%
4117             * Local config file \bbl@opt@config.cfg used^^J%
4118             *}}}%
4119   {\bbl@error{config-not-found}{}}{}%
4120 \fi

```

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

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

```

4121 \def\bbl@tempf{,}
4122 \bbl@foreach\@raw@classoptionslist{%
4123   \in@{=}{#1}%
4124   \ifin@else
4125     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4126   \fi}
4127 \ifx\bbl@opt@main\@nnil
4128   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4129     \let\bbl@tempb\@empty
4130     \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}%
4131     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4132     \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4133       \ifx\bbl@opt@main\@nnil % i.e., if not yet assigned
4134         \ifodd\bbl@iniflag % = *=
4135           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4136         \else % n +=
4137           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4138         \fi
4139       \fi}%
4140   \fi
4141 \else
4142   \bbl@info{Main language set with 'main='. Except if you have\\%
4143     problems, prefer the default mechanism for setting\\%
4144     the main language, i.e., as the last declared.\\%

```

```

4145         Reported}
4146 \fi

```

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

```

4147 \ifx\bbl@opt@main\@nnil\else
4148   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4149   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4150 \fi

```

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

```

4151 \bbl@foreach\bbl@language@opts{%
4152   \def\bbl@tempa{#1}%
4153   \ifx\bbl@tempa\bbl@opt@main\else
4154     \ifnum\bbl@iniflag<\tw@    % 0 0 (other = ldf)
4155       \bbl@ifunset{ds@#1}%
4156       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4157       {}%
4158     \else                      % + * (other = ini)
4159       \DeclareOption{#1}{%
4160         \bbl@ldfinit
4161         \babelprovide[@import]{#1}% %%%
4162         \bbl@afterldf{}}}%
4163   \fi
4164 \fi}
4165 \bbl@foreach\bbl@tempf{%
4166   \def\bbl@tempa{#1}%
4167   \ifx\bbl@tempa\bbl@opt@main\else
4168     \ifnum\bbl@iniflag<\tw@    % 0 0 (other = ldf)
4169     \bbl@ifunset{ds@#1}%
4170     {\IfFileExists{#1.ldf}%
4171      {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4172      {}}%
4173     {}%
4174   \else                      % + * (other = ini)
4175     \IfFileExists{babel-#1.tex}%
4176     {\DeclareOption{#1}{%
4177       \bbl@ldfinit
4178       \babelprovide[@import]{#1}% %%%
4179       \bbl@afterldf{}}}%
4180     {}%
4181   \fi
4182 \fi}

```

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

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

```

4183 \NewHook{babel/presets}
4184 \UseHook{babel/presets}
4185 \def\AfterBabelLanguage#1{%
4186   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4187   \DeclareOption*{}
4188   \ProcessOptions*

```

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

```

4189 \bbl@trace{Option 'main'}

```

```

4190 \ifx\bbl@opt@main\@nnil
4191 \edef\bbl@tempa{\bbl@tempf,\bbl@language@opts}
4192 \let\bbl@tempc\@empty
4193 \edef\bbl@templ{\bbl@loaded,}
4194 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4195 \bbl@for\bbl@tempb\bbl@tempa{%
4196   \edef\bbl@tempd{\bbl@tempb,}%
4197   \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4198   \bbl@xin{\bbl@tempd}{\bbl@templ}%
4199   \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4200 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4201 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4202 \ifx\bbl@tempb\bbl@tempc\else
4203   \bbl@warning{%
4204     Last declared language option is '\bbl@tempc',\%
4205     but the last processed one was '\bbl@tempb'.\%
4206     The main language can't be set as both a global\%
4207     and a package option. Use 'main=\bbl@tempc' as\%
4208     option. Reported}
4209 \fi
4210 \else
4211 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4212   \bbl@ldfinit
4213   \let\CurrentOption\bbl@opt@main
4214   \bbl@exp{% \bbl@opt@provide = empty if *
4215     \\babelprovide
4216     [\bbl@opt@provide,@import,main]% %%%
4217     {\bbl@opt@main}}%
4218   \bbl@afterldf{}
4219   \DeclareOption{\bbl@opt@main}{}
4220 \else % case 0,2 (main is ldf)
4221   \ifx\bbl@loadmain\relax
4222     \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4223   \else
4224     \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4225   \fi
4226   \ExecuteOptions{\bbl@opt@main}
4227   \@namedef{ds@\bbl@opt@main}{}%
4228 \fi
4229 \DeclareOption*{}
4230 \ProcessOptions*
4231 \fi
4232 \bbl@exp{%
4233   \\AtBeginDocument{\\bbl@usehooks@lang{/}{\begindocument}{}}}%
4234 \def\AfterBabelLanguage{\bbl@error{late-after-babel}}{}{}

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

4235 \ifx\bbl@main@language\@undefined
4236   \bbl@info{%
4237     You haven't specified a language as a class or package\%
4238     option. I'll load 'nil'. Reported}
4239   \bbl@load@language{nil}
4240 \fi
4241 </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`

```
4242 \kernel
4243 \let\bbl@onlyswitch\@empty
4244 \input babel.def
4245 \let\bbl@onlyswitch\@undefined
4246 \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.

```
4247 \errors
4248 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4249 \catcode`\:=12 \catcode`\.,=12 \catcode`\.=12 \catcode`\-=12
4250 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4251 \catcode`\@=11 \catcode`\^=7
4252 %
4253 \ifx\MessageBreak\@undefined
4254 \gdef\bbl@error@i#1#2{%
4255 \begingroup
4256 \newlinechar=`^^J
4257 \def\{^^J(babel) }%
4258 \errhelp{#2}\errmessage{\{#1}%
4259 \endgroup}
4260 \else
4261 \gdef\bbl@error@i#1#2{%
4262 \begingroup
4263 \def\{\MessageBreak}%
4264 \PackageError{babel}{#1}{#2}%
4265 \endgroup}
4266 \fi
4267 \def\bbl@errmessage#1#2#3{%
4268 \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4269 \bbl@error@i{#2}{#3}}
4270 % Implicit #2#3#4:
4271 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4272 %
4273 \bbl@errmessage{not-yet-available}
4274 {Not yet available}%
4275 {Find an armchair, sit down and wait}
4276 \bbl@errmessage{bad-package-option}%
4277 {Bad option '#1=#2'. Either you have misspelled the\\%
4278 key or there is a previous setting of '#1'. Valid\\%
4279 keys are, among others, 'shorthands', 'main', 'bidi',\\%
4280 'strings', 'config', 'headfoot', 'safe', 'math'.}%
4281 {See the manual for further details.}
4282 \bbl@errmessage{base-on-the-fly}
4283 {For a language to be defined on the fly 'base'\\%
4284 is not enough, and the whole package must be\\%
4285 loaded. Either delete the 'base' option or\\%
4286 request the languages explicitly}%
4287 {See the manual for further details.}
4288 \bbl@errmessage{undefined-language}
4289 {You haven't defined the language '#1' yet.\\%
```

```

4290     Perhaps you misspelled it or your installation\\%
4291     is not complete}%
4292     {Your command will be ignored, type <return> to proceed}
4293 \bbl@errmessage{shorthand-is-off}
4294     {I can't declare a shorthand turned off (\string#2)}
4295     {Sorry, but you can't use shorthands which have been\\%
4296     turned off in the package options}
4297 \bbl@errmessage{not-a-shorthand}
4298     {The character '\string #1' should be made a shorthand character;\\%
4299     add the command \string\usesshorthands\string{#1\string} to
4300     the preamble.\\%
4301     I will ignore your instruction}%
4302     {You may proceed, but expect unexpected results}
4303 \bbl@errmessage{not-a-shorthand-b}
4304     {I can't switch '\string#2' on or off--not a shorthand}%
4305     {This character is not a shorthand. Maybe you made\\%
4306     a typing mistake? I will ignore your instruction.}
4307 \bbl@errmessage{unknown-attribute}
4308     {The attribute #2 is unknown for language #1.}%
4309     {Your command will be ignored, type <return> to proceed}
4310 \bbl@errmessage{missing-group}
4311     {Missing group for string \string#1}%
4312     {You must assign strings to some category, typically\\%
4313     captions or extras, but you set none}
4314 \bbl@errmessage{only-lua-xe}
4315     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4316     {Consider switching to these engines.}
4317 \bbl@errmessage{only-lua}
4318     {This macro is available only in LuaLaTeX}%
4319     {Consider switching to that engine.}
4320 \bbl@errmessage{unknown-provide-key}
4321     {Unknown key '#1' in \string\babelprovide}%
4322     {See the manual for valid keys}%
4323 \bbl@errmessage{unknown-mapfont}
4324     {Option '\bbl@KVP@mapfont' unknown for\\%
4325     mapfont. Use 'direction'}%
4326     {See the manual for details.}
4327 \bbl@errmessage{no-ini-file}
4328     {There is no ini file for the requested language\\%
4329     (#1: \language). Perhaps you misspelled it or your\\%
4330     installation is not complete}%
4331     {Fix the name or reinstall babel.}
4332 \bbl@errmessage{digits-is-reserved}
4333     {The counter name 'digits' is reserved for mapping\\%
4334     decimal digits}%
4335     {Use another name.}
4336 \bbl@errmessage{limit-two-digits}
4337     {Currently two-digit years are restricted to the\\
4338     range 0-9999}%
4339     {There is little you can do. Sorry.}
4340 \bbl@errmessage{alphabetic-too-large}
4341 {Alphabetic numeral too large (#1)}%
4342 {Currently this is the limit.}
4343 \bbl@errmessage{no-ini-info}
4344     {I've found no info for the current locale.\\%
4345     The corresponding ini file has not been loaded\\%
4346     Perhaps it doesn't exist}%
4347     {See the manual for details.}
4348 \bbl@errmessage{unknown-ini-field}
4349     {Unknown field '#1' in \string\BCPdata.\\%
4350     Perhaps you misspelled it}%
4351     {See the manual for details.}
4352 \bbl@errmessage{unknown-locale-key}

```



```

4353 {Unknown key for locale '#2':\\%
4354 #3\\%
4355 \string#1 will be set to \string\relax}%
4356 {Perhaps you misspelled it.}%
4357 \bbl@errmessage{adjust-only-vertical}
4358 {Currently, #1 related features can be adjusted only\\%
4359 in the main vertical list}%
4360 {Maybe things change in the future, but this is what it is.}
4361 \bbl@errmessage{layout-only-vertical}
4362 {Currently, layout related features can be adjusted only\\%
4363 in vertical mode}%
4364 {Maybe things change in the future, but this is what it is.}
4365 \bbl@errmessage{bidi-only-lua}
4366 {The bidi method 'basic' is available only in\\%
4367 luatex. I'll continue with 'bidi=default', so\\%
4368 expect wrong results}%
4369 {See the manual for further details.}
4370 \bbl@errmessage{multiple-bidi}
4371 {Multiple bidi settings inside a group}%
4372 {I'll insert a new group, but expect wrong results.}
4373 \bbl@errmessage{unknown-package-option}
4374 {Unknown option '\CurrentOption'. Either you misspelled it\\%
4375 or the language definition file \CurrentOption.ldf\\%
4376 was not found%
4377 \bbl@tempa}
4378 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4379 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4380 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4381 \bbl@errmessage{config-not-found}
4382 {Local config file '\bbl@opt@config.cfg' not found}%
4383 {Perhaps you misspelled it.}
4384 \bbl@errmessage{late-after-babel}
4385 {Too late for \string\AfterBabelLanguage}%
4386 {Languages have been loaded, so I can do nothing}
4387 \bbl@errmessage{double-hyphens-class}
4388 {Double hyphens aren't allowed in \string\babelcharclass\\%
4389 because it's potentially ambiguous}%
4390 {See the manual for further info}
4391 \bbl@errmessage{unknown-interchar}
4392 {'#1' for '\language' cannot be enabled.\\%
4393 Maybe there is a typo}%
4394 {See the manual for further details.}
4395 \bbl@errmessage{unknown-interchar-b}
4396 {'#1' for '\language' cannot be disabled.\\%
4397 Maybe there is a typo}%
4398 {See the manual for further details.}
4399 \bbl@errmessage{charproperty-only-vertical}
4400 {\string\babelcharproperty\space can be used only in\\%
4401 vertical mode (preamble or between paragraphs)}%
4402 {See the manual for further info}
4403 \bbl@errmessage{unknown-char-property}
4404 {No property named '#2'. Allowed values are\\%
4405 direction (bc), mirror (bmg), and linebreak (lb)}%
4406 {See the manual for further info}
4407 \bbl@errmessage{bad-transform-option}
4408 {Bad option '#1' in a transform.\\%
4409 I'll ignore it but expect more errors}%
4410 {See the manual for further info.}
4411 \bbl@errmessage{font-conflict-transforms}
4412 {Transforms cannot be re-assigned to different\\%
4413 fonts. The conflict is in '\bbl@kv@label'.\\%
4414 Apply the same fonts or use a different label}%
4415 {See the manual for further details.}

```

```

4416 \bbl@errmessage{transform-not-available}
4417   {'#1' for '\language' cannot be enabled.\\%
4418   Maybe there is a typo or it's a font-dependent transform}%
4419   {See the manual for further details.}
4420 \bbl@errmessage{transform-not-available-b}
4421   {'#1' for '\language' cannot be disabled.\\%
4422   Maybe there is a typo or it's a font-dependent transform}%
4423   {See the manual for further details.}
4424 \bbl@errmessage{year-out-range}
4425   {Year out of range.\\%
4426   The allowed range is #1}%
4427   {See the manual for further details.}
4428 \bbl@errmessage{only-pdfTeX-lang}
4429   {The '#1' ldf style doesn't work with #2,\\%
4430   but you can use the ini locale instead.\\%
4431   Try adding 'provide=' to the option list. You may\\%
4432   also want to set 'bidi=' to some value}%
4433   {See the manual for further details.}
4434 \bbl@errmessage{hyphenmins-args}
4435   {\string\babelhyphenmins\ accepts either the optional\\%
4436   argument or the star, but not both at the same time}%
4437   {See the manual for further details.}
4438 </errors>
4439 <: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.

```

4440 <@Make sure ProvidesFile is defined@>
4441 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4442 \xdef\bbl@format{\jobname}
4443 \def\bbl@version{<@version@>}
4444 \def\bbl@date{<@date@>}
4445 \ifx\AtBeginDocument\@undefined
4446   \def\@empty{}
4447 \fi
4448 <@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.

```

4449 \def\process@line#1#2 #3 #4 {%
4450   \ifx=#1%
4451     \process@synonym{#2}%
4452   \else
4453     \process@language{#1#2}{#3}{#4}%
4454   \fi
4455   \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.

```

4456 \toks@{}
4457 \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.

```

4458 \def\process@synonym#1{%
4459   \ifnum\last@language=\m@ne
4460     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4461   \else
4462     \expandafter\chardef\csname l@#1\endcsname\last@language
4463     \wlog{\string\l@#1=\string\language\the\last@language}%
4464     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4465       \csname\language\hyphenmins\endcsname
4466     \let\bbl@elt\relax
4467     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}}%
4468   \fi}

```

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

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

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

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

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

```

4469 \def\process@language#1#2#3{%
4470   \expandafter\addlanguage\csname l@#1\endcsname
4471   \expandafter\language\csname l@#1\endcsname
4472   \edef\language{#1}%
4473   \bbl@hook@everylanguage{#1}%
4474   % > luatex
4475   \bbl@get@enc#1:.\@@@
4476   \begin{group}
4477     \lefthyphenmin\m@ne
4478     \bbl@hook@loadpatterns{#2}%
4479     % > luatex
4480     \ifnum\lefthyphenmin=\m@ne
4481     \else
4482       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4483         \the\lefthyphenmin\the\righthyphenmin}%
4484     \fi
4485   \endgroup
4486   \def\bbl@tempa{#3}%
4487   \ifx\bbl@tempa\@empty\else
4488     \bbl@hook@loadexceptions{#3}%
4489     % > luatex
4490   \fi
4491   \let\bbl@elt\relax

```

```

4492 \edef\bbl@languages{%
4493   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4494 \ifnum\the\language=\z@
4495   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4496     \set@hyphenmins\tw@thr@@\relax
4497   \else
4498     \expandafter\expandafter\expandafter\set@hyphenmins
4499     \csname #1hyphenmins\endcsname
4500   \fi
4501   \the\toks@
4502   \toks@{}%
4503 \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.

```

4504 \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.

```

4505 \def\bbl@hook@everylanguage#1{}
4506 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4507 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4508 \def\bbl@hook@loadkernel#1{%
4509   \def\addlanguage{\csname newlanguage\endcsname}%
4510   \def\adddialect##1##2{%
4511     \global\chardef##1##2\relax
4512     \wlog{\string##1 = a dialect from \string\language##2}}%
4513   \def\iflanguage##1{%
4514     \expandafter\ifx\csname l@##1\endcsname\relax
4515       \@nolanerr{##1}%
4516     \else
4517       \ifnum\csname l@##1\endcsname=\language
4518         \expandafter\expandafter\expandafter\@firstoftwo
4519       \else
4520         \expandafter\expandafter\expandafter\@secondoftwo
4521       \fi
4522     \fi}%
4523   \def\providehyphenmins##1##2{%
4524     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4525       \@namedef{##1hyphenmins}{##2}%
4526     \fi}%
4527   \def\set@hyphenmins##1##2{%
4528     \lefthyphenmin##1\relax
4529     \righthyphenmin##2\relax}%
4530   \def\selectlanguage{%
4531     \errhelp{Selecting a language requires a package supporting it}%
4532     \errmessage{No multilingual package has been loaded}}%
4533   \let\foreignlanguage\selectlanguage
4534   \let\otherlanguage\selectlanguage
4535   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4536   \def\bbl@usehooks##1##2{% TODO. Temporary!!}
4537   \def\setlocale{%
4538     \errhelp{Find an armchair, sit down and wait}%
4539     \errmessage{(babel) Not yet available}}%
4540   \let\uselocale\setlocale
4541   \let\locale\setlocale
4542   \let\selectlocale\setlocale
4543   \let\localename\setlocale
4544   \let\textlocale\setlocale
4545   \let\textlanguage\setlocale
4546   \let\languagegetext\setlocale}

```

```

4547 \begingroup
4548   \def\AddBabelHook#1#2{%
4549     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4550       \def\next{\toks1}%
4551     \else
4552       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4553     \fi
4554     \next}
4555 \ifx\directlua\@undefined
4556 \ifx\XeTeXinputencoding\@undefined\else
4557   \input xebabel.def
4558 \fi
4559 \else
4560   \input luababel.def
4561 \fi
4562 \openin1 = babel-\bbl@format.cfg
4563 \ifeof1
4564 \else
4565   \input babel-\bbl@format.cfg\relax
4566 \fi
4567 \closein1
4568 \endgroup
4569 \bbl@hook@loadkernel{switch.def}

```

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

```

4570 \openin1 = language.dat

```

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

```

4571 \def\language{english}%
4572 \ifeof1
4573   \message{I couldn't find the file language.dat,\space
4574     I will try the file hyphen.tex}
4575   \input hyphen.tex\relax
4576   \chardef\l@english\z@
4577 \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`.

```

4578   \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.

```

4579   \loop
4580     \endlinechar\m@ne
4581     \read1 to \bbl@line
4582     \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.

```

4583   \if T\ifeof1\fi T\relax
4584   \ifx\bbl@line\@empty\else
4585     \edef\bbl@line{\bbl@line\space\space\space}%
4586     \expandafter\process@line\bbl@line\relax
4587   \fi
4588   \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.

```

4589 \begingroup
4590 \def\bbl@elt#1#2#3#4{%
4591 \global\language=#2\relax
4592 \gdef\language#1}%
4593 \def\bbl@elt##1##2##3##4{}}%
4594 \bbl@languages
4595 \endgroup
4596 \fi
4597 \closein1

```

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

```

4598 \if/\the\toks@/\else
4599 \errhelp{language.dat loads no language, only synonyms}
4600 \errmessage{Orphan language synonym}
4601 \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.

```

4602 \let\bbl@line\@undefined
4603 \let\process@line\@undefined
4604 \let\process@synonym\@undefined
4605 \let\process@language\@undefined
4606 \let\bbl@get@enc\@undefined
4607 \let\bbl@hyph@enc\@undefined
4608 \let\bbl@tempa\@undefined
4609 \let\bbl@hook@loadkernel\@undefined
4610 \let\bbl@hook@everylanguage\@undefined
4611 \let\bbl@hook@loadpatterns\@undefined
4612 \let\bbl@hook@loadexceptions\@undefined
4613 \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).

```

4614 \langle *More package options \rangle ≡
4615 \chardef\bbl@bidimode\z@
4616 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4617 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4618 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4619 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4620 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4621 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4622 \rangle /More package options \rangle

```

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

```

4623 \langle *Font selection \rangle ≡
4624 \bbl@trace{Font handling with fontspec}
4625 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4626 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4627 \DisableBabelHook{babel-fontspec}
4628 \@onlypreamble\bblfont
4629 \newcommand\bblfont[2][ ]{% 1=langs/scripts 2=fam
4630 \ifx\fontspec\@undefined
4631 \usepackage{fontspec}%
4632 \fi

```

```

4633 \EnableBabelHook{babel-fontspec}%
4634 \edef\bbl@tempa{#1}%
4635 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4636 \bbl@bblfont}
4637 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4638 \bbl@ifunset{\bbl@tempb family}%
4639 {\bbl@providfam{\bbl@tempb}}%
4640 {}%
4641 % For the default font, just in case:
4642 \bbl@ifunset{\bbl@sys@\language}\bbl@provide@sys{\language}}{%
4643 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4644 {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
4645 \bbl@exp{%
4646 \let<\bbl@tempb dflt@\language><\bbl@tempb dflt@>%
4647 \\\bbl@fontset<\bbl@tempb dflt@\language>%
4648 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4649 {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4650 \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:

```

4651 \def\bbl@providfam#1{%
4652 \bbl@exp{%
4653 \\\newcommand<#1default>{}% Just define it
4654 \\\bbl@add@list\\bbl@font@fams{#1}%
4655 \\\NewHook{#1family}%
4656 \\\DeclareRobustCommand<#1family>{%
4657 \\\not@math@alphabet<#1family>\relax
4658 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4659 \\\fontfamily<#1default>%
4660 \\\UseHook{#1family}%
4661 \\\selectfont}%
4662 \\\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.

```

4663 \def\bbl@nostdfont#1{%
4664 \bbl@ifunset{\bbl@WFF@f@family}%
4665 {\bbl@csarg\gdef{WFF@f@family}}% Flag, to avoid dupl warns
4666 \bbl@infowarn{The current font is not a babel standard family:\\%
4667 #1%
4668 \fontname\font\\%
4669 There is nothing intrinsically wrong with this warning, and\\%
4670 you can ignore it altogether if you do not need these\\%
4671 families. But if they are used in the document, you should be\\%
4672 aware 'babel' will not set Script and Language for them, so\\%
4673 you may consider defining a new family with \string\babelfont.\\%
4674 See the manual for further details about \string\babelfont.\\%
4675 Reported}}
4676 {}}%
4677 \gdef\bbl@switchfont{%
4678 \bbl@ifunset{\bbl@sys@\language}\bbl@provide@sys{\language}}{%
4679 \bbl@exp{% e.g., Arabic -> arabic
4680 \\\lowercase{\edef\\bbl@tempa{\bbl@c{l}{sname}}}%
4681 \bbl@foreach\bbl@font@fams{%
4682 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4683 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4684 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4685 {}% 123=F - nothing!
4686 {\bbl@exp{% 3=T - from generic
4687 \global\let<\bbl@##1dflt@\language>%
4688 \<\bbl@##1dflt@>}}}%
4689 {\bbl@exp{% 2=T - from script
4690 \global\let<\bbl@##1dflt@\language>%
4691 \<\bbl@##1dflt@*\bbl@tempa>}}}%

```

```

4692     {}}%                                l=T - language, already defined
4693 \def\bbl@tempa{\bbl@nostdfont{}}%      TODO. Don't use \bbl@tempa
4694 \bbl@foreach\bbl@font@fams{%           don't gather with prev for
4695     \bbl@ifunset{\bbl@##1dflt@\language}%
4696     {\bbl@cs{famrst@##1}%
4697     \global\bbl@csarg\let{famrst@##1}\relax}%
4698     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4699         \\bbl@add\\originalTeX{%
4700             \\bbl@font@rst{\bbl@cl{##1dflt}}%
4701                 \<##1default>\<##1family>{##1}}%
4702             \\bbl@font@set{\bbl@##1dflt@\language}% the main part!
4703                 \<##1default>\<##1family>}}}%
4704 \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`.

```

4705 \ifx\f@family\undefined\else          % if latex
4706     \ifcase\bbl@engine                  % if pdftex
4707         \let\bbl@ckeckstdfonts\relax
4708     \else
4709         \def\bbl@ckeckstdfonts{%
4710             \begingroup
4711             \global\let\bbl@ckeckstdfonts\relax
4712             \let\bbl@tempa\empty
4713             \bbl@foreach\bbl@font@fams{%
4714                 \bbl@ifunset{\bbl@##1dflt@}%
4715                 {\@nameuse{##1family}%
4716                 \bbl@csarg\gdef{WFF@f@family}{}% Flag
4717                 \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4718                     \space\space\fontname\font\\}%
4719                 \bbl@csarg\xdef{##1dflt@}{\f@family}%
4720                 \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4721                 {}}%
4722             \ifx\bbl@tempa\empty\else
4723                 \bbl@infowarn{The following font families will use the default\\%
4724                     settings for all or some languages:\\%
4725                     \bbl@tempa
4726                     There is nothing intrinsically wrong with it, but\\%
4727                     'babel' will no set Script and Language, which could\\%
4728                     be relevant in some languages. If your document uses\\%
4729                     these families, consider redefining them with \string\babelfont.\\%
4730                     Reported}%
4731             \fi
4732             \endgroup}
4733 \fi
4734 \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*`).

```

4735 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4736     \bbl@xin@{<>}{#1}%
4737     \ifin@
4738         \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4739     \fi
4740     \bbl@exp{%
4741         \def\\#2{#1}%                'Unprotected' macros return prev values
                                         e.g., \rmdefault{\bbl@rmdflt@lang}

```



```

4742   \\bbl@ifsamestring{#2}{\f@family}%
4743   {\#3%
4744   \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4745   \let\\bbl@tempa\relax}%
4746   {}}}

```

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).

```

4747 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4748 \let\bbl@tempe\bbl@mapselect
4749 \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
4750 \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}}}%
4751 \let\bbl@mapselect\relax
4752 \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
4753 \let#4\@empty % Make sure \renewfontfamily is valid
4754 \bbl@set@renderer
4755 \bbl@exp{%
4756 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% e.g., '\rmfamily '
4757 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4758 {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4759 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4760 {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4761 \\renewfontfamily\\#4%
4762 [\bbl@cl{lsys},% xetex removes unknown features :-(
4763 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4764 #2]}{#3}% i.e., \bbl@exp{..}{#3}
4765 \bbl@unset@renderer
4766 \begingroup
4767 #4%
4768 \xdef#1{\f@family}% e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4769 \endgroup % TODO. Find better tests:
4770 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4771 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4772 \ifin@
4773 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4774 \fi
4775 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4776 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4777 \ifin@
4778 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4779 \fi
4780 \let#4\bbl@temp@fam
4781 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4782 \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.

```

4783 \def\bbl@font@rst#1#2#3#4{%
4784 \bbl@ccarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4785 \def\bbl@font@fams{rm,sf,tt}
4786 <</Font selection>>

```

\BabelFootnote Footnotes.

```

4787 <<{*Footnote changes}>> ≡
4788 \bbl@trace{Bidi footnotes}
4789 \ifnum\bbl@bidimode>\z@ % Any bidi=
4790 \def\bbl@footnote#1#2#3{%

```

```

4791 \@ifnextchar[%
4792   {\bbl@footnote@o{#1}{#2}{#3}}%
4793   {\bbl@footnote@x{#1}{#2}{#3}}}
4794 \long\def\bbl@footnote@x#1#2#3#4{%
4795   \bgroup
4796   \select@language@x{\bbl@main@language}%
4797   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4798   \egroup}
4799 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4800   \bgroup
4801   \select@language@x{\bbl@main@language}%
4802   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4803   \egroup}
4804 \def\bbl@footnotetext#1#2#3{%
4805   \ifnextchar[%
4806     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4807     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4808 \long\def\bbl@footnotetext@x#1#2#3#4{%
4809   \bgroup
4810   \select@language@x{\bbl@main@language}%
4811   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4812   \egroup}
4813 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4814   \bgroup
4815   \select@language@x{\bbl@main@language}%
4816   \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4817   \egroup}
4818 \def\BabelFootnote#1#2#3#4{%
4819   \ifx\bbl@fn@footnote\@undefined
4820     \let\bbl@fn@footnote\footnote
4821   \fi
4822   \ifx\bbl@fn@footnotetext\@undefined
4823     \let\bbl@fn@footnotetext\footnotetext
4824   \fi
4825   \bbl@ifblank{#2}%
4826     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4827     \@namedef{\bbl@stripslash#1text}%
4828     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4829   {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4830   \@namedef{\bbl@stripslash#1text}%
4831   {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4832 \fi
4833 <</Footnote changes>>

```

10. Hooks for XeTeX and LuaTeX

10.1. XeTeX

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

Now, the code.

```

4834 <{*xetex}
4835 \def\BabelStringsDefault{unicode}
4836 \let\xebbl@stop\relax
4837 \AddBabelHook{xetex}{encodedcommands}{%
4838   \def\bbl@tempa{#1}%
4839   \ifx\bbl@tempa\@empty
4840     \XeTeXinputencoding"bytes"%
4841   \else
4842     \XeTeXinputencoding"#1"%
4843   \fi
4844   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}

```

```

4845 \AddBabelHook{xetex}{stopcommands}{%
4846   \xebbl@stop
4847   \letxebbl@stop\relax}
4848 \def\bbl@input@classes{% Used in CJK intraspaces
4849   \input{load-unicode-xetex-classes.tex}%
4850   \let\bbl@input@classes\relax}
4851 \def\bbl@intraspace#1 #2 #3\@@{%
4852   \bbl@csarg\gdef{xeisp@\language\language}%
4853     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4854 \def\bbl@intrapenalty#1\@@{%
4855   \bbl@csarg\gdef{xeipn@\language\language}%
4856     {\XeTeXlinebreakpenalty #1\relax}}
4857 \def\bbl@provide@intraspace{%
4858   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}}%
4859   \ifin\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4860   \ifin@
4861     \bbl@ifunset{bbl@intsp@\language\language}{}%
4862     {\expandafter\ifx\csname bbl@intsp@\language\language\endcsname\@empty\else
4863       \ifx\bbl@KVP@intraspace\@nnil
4864         \bbl@exp{%
4865           \\bbl@intraspace\bbl@cl{intsp}\\}\@@}%
4866         \fi
4867         \ifx\bbl@KVP@intrapenalty\@nnil
4868           \bbl@intrapenalty0\@@
4869           \fi
4870         \fi
4871         \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4872           \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4873           \fi
4874           \ifx\bbl@KVP@intrapenalty\@nnil\else
4875             \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4876             \fi
4877             \bbl@exp{%
4878               % TODO. Execute only once (but redundant):
4879               \\bbl@add\<extras\language\>{%
4880                 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4881                 \<bbl@xeisp@\language\>%
4882                 \<bbl@xeipn@\language\>%
4883                 \\bbl@toglobal\<extras\language\>%
4884                 \\bbl@add\<noextras\language\>{%
4885                   \XeTeXlinebreaklocale ""}%
4886                   \\bbl@toglobal\<noextras\language\>%
4887                   \ifx\bbl@ispacesize\@undefined
4888                     \gdef\bbl@ispacesize{\bbl@cl{xeisp}}}%
4889                     \ifx\AtBeginDocument\@notprerr
4890                       \expandafter\@secondoftwo % to execute right now
4891                       \fi
4892                       \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4893                       \fi}%
4894     \fi}
4895 \ifx\DisableBabelHook\@undefined\endinput\fi %%% TODO: why
4896 \let\bbl@set@renderer\relax
4897 \let\bbl@unset@renderer\relax
4898 <@Font selection@>
4899 \def\bbl@provide@extra#1{}

```

10.2. Support for interchar

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

```

4900 \ifnum\Xe@alloc@intercharclass<\thr@@
4901   \Xe@alloc@intercharclass\thr@@
4902 \fi

```

```

4903 \chardef\bbl@xeclasse@default@=\z@
4904 \chardef\bbl@xeclasse@ckideogram@=\@ne
4905 \chardef\bbl@xeclasse@ckleftpunctuation@=\tw@
4906 \chardef\bbl@xeclasse@ckrightpunctuation@=\thr@@
4907 \chardef\bbl@xeclasse@boundary@=4095
4908 \chardef\bbl@xeclasse@ignore@=4096

```

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

```

4909 \AddBabelHook{babel-interchar}{beforeextras}{%
4910   \@nameuse{bbl@xechars@\language@}}
4911 \DisableBabelHook{babel-interchar}
4912 \protected\def\bbl@charclass#1{%
4913   \ifnum\count@<\z@
4914     \count@=-\count@
4915     \loop
4916       \bbl@exp{%
4917         \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4918         \XeTeXcharclass\count@ \bbl@tempc
4919         \ifnum\count@<`#1\relax
4920         \advance\count@\@ne
4921       \repeat
4922   \else
4923     \babel@savevariable{\XeTeXcharclass`#1}%
4924     \XeTeXcharclass`#1 \bbl@tempc
4925   \fi
4926   \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@usingxeclasse\bbl@xeclasse@punct@english\bbl@charclass{.}` `\bbl@charclass{,}` (etc.), where `\bbl@usingxeclasse` 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.

```

4927 \newcommand\bbl@ifinterchar[1]{%
4928   \let\bbl@tempa\@gobble      % Assume to ignore
4929   \edef\bbl@tempb{\zap@space#1 \@empty}%
4930   \ifx\bbl@KVP@interchar\@nnil\else
4931     \bbl@replace\bbl@KVP@interchar{ }{,}%
4932     \bbl@foreach\bbl@tempb{%
4933       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
4934     \ifin@
4935       \let\bbl@tempa\@firstofone
4936     \fi}%
4937   \fi
4938   \bbl@tempa}
4939 \newcommand\IfBabelIntercharT[2]{%
4940   \bbl@carg\bbl@add{bbl@icsave@\CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
4941 \newcommand\babelcharclass[3]{%
4942   \EnableBabelHook{babel-interchar}%
4943   \bbl@csarg\newXeTeXintercharclass{xeclasse@#2@#1}%
4944   \def\bbl@tempb##1{%
4945     \ifx##1\@empty\else
4946       \ifx##1-
4947         \bbl@upto
4948       \else
4949         \bbl@charclass{%
4950           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4951       \fi
4952       \expandafter\bbl@tempb
4953     \fi}%
4954   \bbl@ifunset{bbl@xechars@#1}%

```

```

4955 {\toks@{%
4956 \babel@savevariable\XeTeXinterchartokenstate
4957 \XeTeXinterchartokenstate\@ne
4958 }}%
4959 {\toks@\expandafter\expandafter\expandafter{%
4960 \csname bbl@xechars@#1\endcsname}}%
4961 \bbl@csarg\edef{xechars@#1}{%
4962 \the\toks@
4963 \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
4964 \bbl@tempb#3\@empty}}
4965 \protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
4966 \protected\def\bbl@upto{%
4967 \ifnum\count@>\z@
4968 \advance\count@\@ne
4969 \count@-\count@
4970 \else\ifnum\count@=\z@
4971 \bbl@charclass{-}%
4972 \else
4973 \bbl@error{double-hyphens-class}{\}\}\}%
4974 \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⟩`.

```

4975 \def\bbl@ignoreinterchar{%
4976 \ifnum\language=\l@nohyphenation
4977 \expandafter\@gobble
4978 \else
4979 \expandafter\@firstofone
4980 \fi}
4981 \newcommand\babelinterchar[5][]{%
4982 \let\bbl@kv@label\@empty
4983 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
4984 \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4985 {\bbl@ignoreinterchar{#5}}%
4986 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
4987 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
4988 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
4989 \XeTeXinterchartoks
4990 \@nameuse{bbl@xeclass@\bbl@tempa @#2}
4991 \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{\}\}\} %
4992 \@nameuse{bbl@xeclass@\bbl@tempb @#2}
4993 \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{\}\}\} %
4994 = \expandafter{%
4995 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
4996 \csname\zap@space bbl@xeinter@\bbl@kv@label
4997 @#3@#4@#2 \@empty\endcsname}}}}
4998 \DeclareRobustCommand\enablelocaleinterchar[1]{%
4999 \bbl@ifunset{bbl@ic@#1@language}%
5000 {\bbl@error{unknown-interchar}{#1}{\}\}\}%
5001 {\bbl@csarg\let{ic@#1@language}\@firstofone}}
5002 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5003 \bbl@ifunset{bbl@ic@#1@language}%
5004 {\bbl@error{unknown-interchar-b}{#1}{\}\}\}%
5005 {\bbl@csarg\let{ic@#1@language}\@gobble}}
5006 /xetex

```

10.3. Layout

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

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

\advance\bbbl@startskip\adim,\bbbl@startskip\adim.

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

```

5007 (*xetex | texxet)
5008 \providecommand\bbbl@provide@intraspace{}
5009 \bbbl@trace{Redefinitions for bidi layout}
5010 \ifx\bbbl@opt@layout\@nnil\else % if layout=..
5011 \def\bbbl@startskip{\ifcase\bbbl@thepardir\leftskip\else\rightskip\fi}
5012 \def\bbbl@endskip{\ifcase\bbbl@thepardir\rightskip\else\leftskip\fi}
5013 \ifnum\bbbl@bidimode>\z@ % TODO: always?
5014 \def\@hangfrom#1{%
5015   \setbox\@tempboxa\hbox{#1}%
5016   \hangindent\ifcase\bbbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5017   \noindent\box\@tempboxa}
5018 \def\raggedright{%
5019   \let\\\@centercr
5020   \bbbl@startskip\z@skip
5021   \@rightskip\@flushglue
5022   \bbbl@endskip\@rightskip
5023   \parindent\z@
5024   \parfillskip\bbbl@startskip}
5025 \def\raggedleft{%
5026   \let\\\@centercr
5027   \bbbl@startskip\@flushglue
5028   \bbbl@endskip\z@skip
5029   \parindent\z@
5030   \parfillskip\bbbl@endskip}
5031 \fi
5032 \IfBabelLayout{lists}
5033 {\bbbl@sreplace\list
5034   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbbl@listleftmargin}%
5035   \def\bbbl@listleftmargin{%
5036     \ifcase\bbbl@thepardir\leftmargin\else\rightmargin\fi}%
5037   \ifcase\bbbl@engine
5038     \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5039     \def\p@enumiii{\p@enumii}\theenumii}%
5040   \fi
5041   \bbbl@sreplace\@verbatim
5042     {\leftskip\@totalleftmargin}%
5043     {\bbbl@startskip\textwidth
5044       \advance\bbbl@startskip-\linewidth}%
5045   \bbbl@sreplace\@verbatim
5046     {\rightskip\z@skip}%
5047     {\bbbl@endskip\z@skip}}%
5048 {}
5049 \IfBabelLayout{contents}
5050 {\bbbl@sreplace\@dottedtocline{\leftskip}{\bbbl@startskip}%
5051   \bbbl@sreplace\@dottedtocline{\rightskip}{\bbbl@endskip}}
5052 {}
5053 \IfBabelLayout{columns}
5054 {\bbbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbbl@outputbox}%
5055   \def\bbbl@outputbox#1{%
5056     \hb@xt@\textwidth{%
5057       \hskip\columnwidth
5058       \hfil
5059       {\normalcolor\vrule \@width\columnseprule}%
5060       \hfil
5061       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5062       \hskip-\textwidth
5063       \hb@xt@\columnwidth{\box\@outputbox \hss}%
5064       \hskip\columnsep
5065       \hskip\columnwidth}}}%
5066 {}

```

```

5067 <@Footnote changes>
5068 \IfBabelLayout{footnotes}%
5069   {\BabelFootnote\footnote\language\language{}{}}%
5070   \BabelFootnote\localfootnote\language\language{}{}}%
5071   \BabelFootnote\mainfootnote{}{}{}%
5072 {}

```

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

```

5073 \IfBabelLayout{counters*}%
5074   {\bbl@add\bbl@opt@layout{.counters.}%
5075     \AddToHook{shipout/before}{%
5076       \let\bbl@tempa\babelsublr
5077       \let\babelsublr\@firstofone
5078       \let\bbl@save@thepage\thepage
5079       \protected@edef\thepage{\thepage}%
5080       \let\babelsublr\bbl@tempa}%
5081     \AddToHook{shipout/after}{%
5082       \let\thepage\bbl@save@thepage}}{}
5083 \IfBabelLayout{counters}%
5084   {\let\bbl@latinarabic=\@arabic
5085     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5086     \let\bbl@asciroman=\@roman
5087     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5088     \let\bbl@asciiRoman=\@Roman
5089     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5090 \fi % end if layout
5091 </xetex | texet>

```

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).

```

5092 <*texet>
5093 \def\bbl@provide@extra#1{%
5094   % == auto-select encoding ==
5095   \ifx\bbl@encoding@select@off\@empty\else
5096     \bbl@ifunset{\bbl@encoding@#1}%
5097     {\def\@elt##1{,##1,}%
5098       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5099       \count@z@
5100       \bbl@foreach\bbl@tempe{%
5101         \def\bbl@tempd{##1}% Save last declared
5102         \advance\count@\@ne}%
5103       \ifnum\count@>\@ne % (1)
5104         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5105         \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5106         \bbl@replace\bbl@tempa{ },}%
5107         \global\bbl@csarg\let{encoding@#1}\@empty
5108         \bbl@xin@{,\bbl@tempd,},{,\bbl@tempa,}%
5109         \ifin@else % if main encoding included in ini, do nothing
5110           \let\bbl@tempb\relax
5111           \bbl@foreach\bbl@tempa{%
5112             \ifx\bbl@tempb\relax
5113               \bbl@xin@{,##1,},{,\bbl@tempe,}%
5114               \ifin@\def\bbl@tempb{##1}\fi
5115             \fi}%
5116           \ifx\bbl@tempb\relax\else
5117             \bbl@exp{%
5118               \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>%
5119               \gdef\<bbl@encoding@#1>{%
5120                 \\\babel@save\\f@encoding
5121                 \\\bbl@add\\originalTeX{\\selectfont}%

```

```

5122          \\\fontencoding{\bbl@tempb}%
5123          \\\selectfont}}%
5124      \fi
5125      \fi
5126      \fi}%
5127      {}%
5128      \fi}
5129 </texxet>

```

10.5. LuaTeX

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

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

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

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

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

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

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

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

```

5130 <*\luatex>
5131 \directlua{ Babel = Babel or {} } % DL2
5132 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5133 \bbl@trace{Read language.dat}
5134 \ifx\bbl@readstream\undefined
5135   \csname newread\endcsname\bbl@readstream
5136 \fi
5137 \begingroup
5138   \toks@{}
5139   \count@ \z@ % 0=start, 1=0th, 2=normal
5140   \def\bbl@process@line#1#2 #3 #4 {%
5141     \ifx=#1%
5142       \bbl@process@synonym{#2}%
5143     \else
5144       \bbl@process@language{#1#2}{#3}{#4}%
5145     \fi
5146     \ignorespaces}
5147   \def\bbl@manylang{%
5148     \ifnum\bbl@last>\@ne
5149       \bbl@info{Non-standard hyphenation setup}%

```



```

5150 \fi
5151 \let\bbl@manylang\relax}
5152 \def\bbl@process@language#1#2#3{%
5153 \ifcase\count@
5154 \ifundefined{zth@#1}{\count@tw@}{\count@ne}%
5155 \or
5156 \count@tw@
5157 \fi
5158 \ifnum\count@=tw@
5159 \expandafter\addlanguage\csname l@#1\endcsname
5160 \language\allocationnumber
5161 \chardef\bbl@last\allocationnumber
5162 \bbl@manylang
5163 \let\bbl@elt\relax
5164 \xdef\bbl@languages{%
5165 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5166 \fi
5167 \the\toks@
5168 \toks@{}}
5169 \def\bbl@process@synonym@aux#1#2{%
5170 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5171 \let\bbl@elt\relax
5172 \xdef\bbl@languages{%
5173 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5174 \def\bbl@process@synonym#1{%
5175 \ifcase\count@
5176 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5177 \or
5178 \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5179 \else
5180 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5181 \fi}
5182 \ifx\bbl@languages@undefined % Just a (sensible?) guess
5183 \chardef\l@english\z@
5184 \chardef\l@USenglish\z@
5185 \chardef\bbl@last\z@
5186 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5187 \gdef\bbl@languages{%
5188 \bbl@elt{english}{0}{hyphen.tex}}%
5189 \bbl@elt{USenglish}{0}{}}
5190 \else
5191 \global\let\bbl@languages@format\bbl@languages
5192 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5193 \ifnum#2>\z@
5194 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5195 \fi}%
5196 \xdef\bbl@languages{\bbl@languages}%
5197 \fi
5198 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5199 \bbl@languages
5200 \openin\bbl@readstream=language.dat
5201 \ifeof\bbl@readstream
5202 \bbl@warning{I couldn't find language.dat. No additional\\%
5203 patterns loaded. Reported}%
5204 \else
5205 \loop
5206 \endlinechar\m@ne
5207 \read\bbl@readstream to \bbl@line
5208 \endlinechar\^^M
5209 \if T\ifeof\bbl@readstream F\fi T\relax
5210 \ifx\bbl@line\empty\else
5211 \edef\bbl@line{\bbl@line\space\space\space}%
5212 \expandafter\bbl@process@line\bbl@line\relax

```

```

5213     \fi
5214   \repeat
5215   \fi
5216   \closein\bbl@readstream
5217 \endgroup
5218 \bbl@trace{Macros for reading patterns files}
5219 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5220 \ifx\babelcatcodetablenum\@undefined
5221   \ifx\newcatcodetable\@undefined
5222     \def\babelcatcodetablenum{5211}
5223     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5224   \else
5225     \newcatcodetable\babelcatcodetablenum
5226     \newcatcodetable\bbl@pattcodes
5227   \fi
5228 \else
5229   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5230 \fi
5231 \def\bbl@luapatterns#1#2{%
5232   \bbl@get@enc#1:.\@@@
5233   \setbox\z@\hbox\bgroup
5234     \begingroup
5235       \savecatcodetable\babelcatcodetablenum\relax
5236       \initcatcodetable\bbl@pattcodes\relax
5237       \catcodetable\bbl@pattcodes\relax
5238       \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5239       \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5240       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5241       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5242       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5243       \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5244       \input #1\relax
5245       \catcodetable\babelcatcodetablenum\relax
5246     \endgroup
5247   \def\bbl@tempa{#2}%
5248   \ifx\bbl@tempa\@empty\else
5249     \input #2\relax
5250   \fi
5251 \egroup}%
5252 \def\bbl@patterns@lua#1{%
5253   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5254     \csname l@#1\endcsname
5255     \edef\bbl@tempa{#1}%
5256   \else
5257     \csname l@#1:\f@encoding\endcsname
5258     \edef\bbl@tempa{#1:\f@encoding}%
5259   \fi\relax
5260   \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5261   \@ifundefined{bbl@hyphendata@the\language}%
5262     {\def\bbl@elt##1##2##3##4{%
5263       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5264         \def\bbl@tempb{##3}%
5265         \ifx\bbl@tempb\@empty\else % if not a synonymous
5266           \def\bbl@tempc{##3}{##4}%
5267         \fi
5268         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5269       \fi}%
5270   \bbl@languages
5271   \@ifundefined{bbl@hyphendata@the\language}%
5272     {\bbl@info{No hyphenation patterns were set for\%
5273       language '\bbl@tempa'. Reported}}%
5274     {\expandafter\expandafter\expandafter\bbl@luapatterns
5275       \csname bbl@hyphendata@the\language\endcsname}}}%

```

5276 \endinput\fi

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

```

5277 \ifx\DisableBabelHook\@undefined
5278   \AddBabelHook{luatex}{everylanguage}{%
5279     \def\process@language##1##2##3{%
5280       \def\process@line####1####2 ####3 ####4 {}}}
5281   \AddBabelHook{luatex}{loadpatterns}{%
5282     \input #1\relax
5283     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5284       {{#1}}}}
5285   \AddBabelHook{luatex}{loadexceptions}{%
5286     \input #1\relax
5287     \def\bbl@tempb##1##2{{##1}{##1}}%
5288     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5289       {\expandafter\expandafter\expandafter\bbl@tempb
5290         \csname bbl@hyphendata@the\language\endcsname}}
5291 \endinput\fi

```

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

```

5292 \begingroup % TODO - to a lua file % DL3
5293 \catcode`\%=12
5294 \catcode`\'=12
5295 \catcode`\|=12
5296 \catcode`\:=12
5297 \directlua{
5298   Babel.locale_props = Babel.locale_props or {}
5299   function Babel.lua_error(e, a)
5300     tex.print([[noexpand\csname bbl@error\endcsname{]] ..
5301       e .. '}' .. (a or '') .. '}{}}')
5302   end
5303   function Babel.bytes(line)
5304     return line:gsub("(.)",
5305       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5306   end
5307   function Babel.begin_process_input()
5308     if luatexbase and luatexbase.add_to_callback then
5309       luatexbase.add_to_callback('process_input_buffer',
5310         Babel.bytes, 'Babel.bytes')
5311     else
5312       Babel.callback = callback.find('process_input_buffer')
5313       callback.register('process_input_buffer', Babel.bytes)
5314     end
5315   end
5316   function Babel.end_process_input ()
5317     if luatexbase and luatexbase.remove_from_callback then
5318       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5319     else
5320       callback.register('process_input_buffer', Babel.callback)
5321     end
5322   end
5323   function Babel.str_to_nodes(fn, matches, base)
5324     local n, head, last
5325     if fn == nil then return nil end
5326     for s in string.utfvalues(fn(matches)) do
5327       if base.id == 7 then
5328         base = base.replace
5329       end
5330       n = node.copy(base)
5331       n.char = s
5332       if not head then
5333         head = n
5334       else

```

```

5335         last.next = n
5336     end
5337     last = n
5338 end
5339 return head
5340 end
5341 Babel.linebreaking = Babel.linebreaking or {}
5342 Babel.linebreaking.before = {}
5343 Babel.linebreaking.after = {}
5344 Babel.locale = {}
5345 function Babel.linebreaking.add_before(func, pos)
5346     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5347     if pos == nil then
5348         table.insert(Babel.linebreaking.before, func)
5349     else
5350         table.insert(Babel.linebreaking.before, pos, func)
5351     end
5352 end
5353 function Babel.linebreaking.add_after(func)
5354     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5355     table.insert(Babel.linebreaking.after, func)
5356 end
5357 function Babel.addpatterns(pp, lg)
5358     local lg = lang.new(lg)
5359     local pats = lang.patterns(lg) or ''
5360     lang.clear_patterns(lg)
5361     for p in pp:gmatch('[^%s]+') do
5362         ss = ''
5363         for i in string.utfcharacters(p:gsub('%d', '')) do
5364             ss = ss .. '%d?' .. i
5365         end
5366         ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5367         ss = ss:gsub('%.%d%?$', '%%.')
5368         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5369         if n == 0 then
5370             tex.sprint(
5371                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5372                 .. p .. [[]])
5373             pats = pats .. ' ' .. p
5374         else
5375             tex.sprint(
5376                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5377                 .. p .. [[]])
5378         end
5379     end
5380     lang.patterns(lg, pats)
5381 end
5382 Babel.characters = Babel.characters or {}
5383 Babel.ranges = Babel.ranges or {}
5384 function Babel.hlist_has_bidi(head)
5385     local has_bidi = false
5386     local ranges = Babel.ranges
5387     for item in node.traverse(head) do
5388         if item.id == node.id'glyph' then
5389             local itemchar = item.char
5390             local chardata = Babel.characters[itemchar]
5391             local dir = chardata and chardata.d or nil
5392             if not dir then
5393                 for nn, et in ipairs(ranges) do
5394                     if itemchar < et[1] then
5395                         break
5396                     elseif itemchar <= et[2] then
5397                         dir = et[3]

```

```

5398         break
5399     end
5400 end
5401 end
5402 if dir and (dir == 'al' or dir == 'r') then
5403     has_bidi = true
5404 end
5405 end
5406 end
5407 return has_bidi
5408 end
5409 function Babel.set_chranges_b (script, chrng)
5410     if chrng == '' then return end
5411     texio.write('Replacing ' .. script .. ' script ranges')
5412     Babel.script_blocks[script] = {}
5413     for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5414         table.insert(
5415             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5416     end
5417 end
5418 function Babel.discard_sublr(str)
5419     if str:find( [[\string\indexentry]] ) and
5420        str:find( [[\string\babelsublr]] ) then
5421         str = str:gsub( [[\string\babelsublr%*{%b{}}]],
5422                        function(m) return m:sub(2,-2) end )
5423     end
5424     return str
5425 end
5426 }
5427 \endgroup
5428 \ifx\newattribute\undefined\else % Test for plain
5429 \newattribute\bbl@attr@locale % DL4
5430 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5431 \AddBabelHook{luatex}{beforeextras}{%
5432     \setattribute\bbl@attr@locale\localeid}
5433 \fi
5434 \def\BabelStringsDefault{unicode}
5435 \let\luabbl@stop\relax
5436 \AddBabelHook{luatex}{encodedcommands}{%
5437     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5438     \ifx\bbl@tempa\bbl@tempb\else
5439         \directlua{Babel.begin_process_input()}%
5440         \def\luabbl@stop{%
5441             \directlua{Babel.end_process_input()}}%
5442     \fi}%
5443 \AddBabelHook{luatex}{stopcommands}{%
5444     \luabbl@stop
5445     \let\luabbl@stop\relax}
5446 \AddBabelHook{luatex}{patterns}{%
5447     \ifundefined{bbl@hyphendata@the\language}%
5448     {\def\bbl@elt##1##2##3##4{%
5449         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5450         \def\bbl@tempb{##3}%
5451         \ifx\bbl@tempb\empty\else % if not a synonymous
5452             \def\bbl@tempc{{##3}{##4}}%
5453         \fi
5454         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5455         \fi}%
5456     \bbl@languages
5457     \ifundefined{bbl@hyphendata@the\language}%
5458     {\bbl@info{No hyphenation patterns were set for\\%
5459         language '#2'. Reported}}%
5460     {\expandafter\expandafter\expandafter\bbl@luapatterns

```

```

5461         \csname bbl@hyphendata@the\language\endcsname\}}}%
5462 \ifundefined{bbl@patterns@}\}%
5463 \beginingroup
5464   \bbl@xin@{,\number\language,}\bbl@pttnlist}%
5465   \ifin@else
5466     \ifx\bbl@patterns@\empty\else
5467       \directlua{ Babel.addpatterns(
5468         [[\bbl@patterns@]], \number\language) }%
5469     \fi
5470     \ifundefined{bbl@patterns@#1}%
5471       \empty
5472       {\directlua{ Babel.addpatterns(
5473         [[\space\csname bbl@patterns@#1\endcsname]],
5474         \number\language) }}%
5475     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5476   \fi
5477 \endgroup}%
5478 \bbl@exp{%
5479   \bbl@ifunset{bbl@prehc@\languagename}\}%
5480   {\bbl@ifblank{\bbl@cs{prehc@\languagename}\}}}%
5481   {\prehyphenchar=\bbl@cl{prehc}\relax}}

```

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

```

5482 \@onlypreamble\babelpatterns
5483 \AtEndOfPackage{%
5484   \newcommand\babelpatterns[2][\empty]{%
5485     \ifx\bbl@patterns@\relax
5486       \let\bbl@patterns@\empty
5487     \fi
5488     \ifx\bbl@pttnlist@\empty\else
5489       \bbl@warning{%
5490         You must not intermingle \string\selectlanguage\space and\%
5491         \string\babelpatterns\space or some patterns will not\%
5492         be taken into account. Reported}%
5493       \fi
5494       \ifx@\empty#1%
5495         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5496       \else
5497         \edef\bbl@tempb{\zap@space#1 \empty}%
5498         \bbl@for\bbl@tempa\bbl@tempb{%
5499           \bbl@fixname\bbl@tempa
5500           \bbl@iflanguage\bbl@tempa{%
5501             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5502               \ifundefined{bbl@patterns@\bbl@tempa}%
5503                 \empty
5504                 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5505               #2}}}%
5506         \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.

```

5507 \def\bbl@intraspace#1 #2 #3\@{
5508   \directlua{
5509     Babel.intraspaces = Babel.intraspaces or {}
5510     Babel.intraspaces['\csname bbl@sbcpr@\languagename\endcsname'] = %
5511     {b = #1, p = #2, m = #3}

```

```

5512     Babel.locale_props[\the\localeid].intraspace = %
5513     {b = #1, p = #2, m = #3}
5514 }
5515 \def\bbl@intrapenalty#1\@@{%
5516   \directlua{
5517     Babel.intrapenalties = Babel.intrapenalties or {}
5518     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5519     Babel.locale_props[\the\localeid].intrapenalty = #1
5520   }
5521 \beginingroup
5522 \catcode`\%=12
5523 \catcode`\&=14
5524 \catcode`\'=12
5525 \catcode`\~ =12
5526 \gdef\bbl@seaintraspace{&
5527   \let\bbl@seaintraspace\relax
5528   \directlua{
5529     Babel.sea_enabled = true
5530     Babel.sea_ranges = Babel.sea_ranges or {}
5531     function Babel.set_chranges (script, chrng)
5532       local c = 0
5533       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5534         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5535         c = c + 1
5536       end
5537     end
5538     function Babel.sea_disc_to_space (head)
5539       local sea_ranges = Babel.sea_ranges
5540       local last_char = nil
5541       local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5542       for item in node.traverse(head) do
5543         local i = item.id
5544         if i == node.id'glyph' then
5545           last_char = item
5546         elseif i == 7 and item.subtype == 3 and last_char
5547           and last_char.char > 0x0C99 then
5548           quad = font.getfont(last_char.font).size
5549           for lg, rg in pairs(sea_ranges) do
5550             if last_char.char > rg[1] and last_char.char < rg[2] then
5551               lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5552               local intraspace = Babel.intraspaces[lg]
5553               local intrapenalty = Babel.intrapenalties[lg]
5554               local n
5555               if intrapenalty ~= 0 then
5556                 n = node.new(14, 0)      &% penalty
5557                 n.penalty = intrapenalty
5558                 node.insert_before(head, item, n)
5559               end
5560               n = node.new(12, 13)      &% (glue, spaceskip)
5561               node.setglue(n, intraspace.b * quad,
5562                 intraspace.p * quad,
5563                 intraspace.m * quad)
5564               node.insert_before(head, item, n)
5565               node.remove(head, item)
5566             end
5567           end
5568         end
5569       end
5570     end
5571   }&
5572   \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.

```
5573 \catcode`\%=14
5574 \gdef\bbl@cjkintraspacespace{%
5575   \let\bbl@cjkintraspacespace\relax
5576   \directlua{
5577     require('babel-data-cjk.lua')
5578     Babel.cjk_enabled = true
5579     function Babel.cjk_linebreak(head)
5580       local GLYPH = node.id'glyph'
5581       local last_char = nil
5582       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5583       local last_class = nil
5584       local last_lang = nil
5585
5586       for item in node.traverse(head) do
5587         if item.id == GLYPH then
5588
5589           local lang = item.lang
5590
5591           local LOCALE = node.get_attribute(item,
5592             Babel.attr_locale)
5593           local props = Babel.locale_props[LOCALE] or {}
5594
5595           local class = Babel.cjk_class[item.char].c
5596
5597           if props.cjk_quotes and props.cjk_quotes[item.char] then
5598             class = props.cjk_quotes[item.char]
5599           end
5600
5601           if class == 'cp' then class = 'cl' % ]) as CL
5602           elseif class == 'id' then class = 'I'
5603           elseif class == 'cj' then class = 'I' % loose
5604           end
5605
5606           local br = 0
5607           if class and last_class and Babel.cjk_breaks[last_class][class] then
5608             br = Babel.cjk_breaks[last_class][class]
5609           end
5610
5611           if br == 1 and props.linebreak == 'c' and
5612             lang ~= \the\l@nohyphenation\space and
5613             last_lang ~= \the\l@nohyphenation then
5614             local intrapenalty = props.intrapenalty
5615             if intrapenalty ~= 0 then
5616               local n = node.new(14, 0)      % penalty
5617               n.penalty = intrapenalty
5618               node.insert_before(head, item, n)
5619             end
5620             local intraspacespace = props.intraspacespace
5621             local n = node.new(12, 13)      % (glue, spaceskip)
5622             node.setglue(n, intraspacespace.b * quad,
5623               intraspacespace.p * quad,
5624               intraspacespace.m * quad)
5625             node.insert_before(head, item, n)
5626           end
5627         end
5628       end
5629     end
5630   }
5631 }
```



```

5628         if font.getfont(item.font) then
5629             quad = font.getfont(item.font).size
5630         end
5631         last_class = class
5632         last_lang = lang
5633     else % if penalty, glue or anything else
5634         last_class = nil
5635     end
5636 end
5637 lang.hyphenate(head)
5638 end
5639 }%
5640 \bbl@luahyphenate}
5641 \gdef\bbl@luahyphenate{%
5642 \let\bbl@luahyphenate\relax
5643 \directlua{
5644     luatexbase.add_to_callback('hyphenate',
5645     function (head, tail)
5646         if Babel.linebreaking.before then
5647             for k, func in ipairs(Babel.linebreaking.before) do
5648                 func(head)
5649             end
5650         end
5651         lang.hyphenate(head)
5652         if Babel.cjk_enabled then
5653             Babel.cjk_linebreak(head)
5654         end
5655         if Babel.linebreaking.after then
5656             for k, func in ipairs(Babel.linebreaking.after) do
5657                 func(head)
5658             end
5659         end
5660         if Babel.set_hboxed then
5661             Babel.set_hboxed(head)
5662         end
5663         if Babel.sea_enabled then
5664             Babel.sea_disc_to_space(head)
5665         end
5666     end,
5667     'Babel.hyphenate')
5668 }}
5669 \endgroup
5670 \def\bbl@provide@intraspace{%
5671 \bbl@ifunset\bbl@intsp@{language}{}%
5672 {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5673 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5674 \ifin@ % cjk
5675 \bbl@cjk_intraspace
5676 \directlua{
5677     Babel.locale_props = Babel.locale_props or {}
5678     Babel.locale_props[\the\localeid].linebreak = 'c'
5679 }%
5680 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5681 \ifx\bbl@KVP@intrapenalty\@nnil
5682 \bbl@intrapenalty0\@%
5683 \fi
5684 \else % sea
5685 \bbl@sea_intraspace
5686 \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@}%
5687 \directlua{
5688     Babel.sea_ranges = Babel.sea_ranges or {}
5689     Babel.set_chranges('\bbl@cl{sbcpr}',
5690         '\bbl@cl{chrng}')

```

```

5691      }%
5692      \ifx\bbl@KVP@intrapenalty\@nnil
5693      \bbl@intrapenalty0\@@
5694      \fi
5695      \fi
5696      \fi
5697      \ifx\bbl@KVP@intrapenalty\@nnil\else
5698      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5699      \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.

```

5700 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5701 \def\bblar@chars{%
5702   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5703   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5704   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5705 \def\bblar@elongated{%
5706   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5707   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5708   0649,064A}
5709 \begingroup
5710   \catcode\_ =11 \catcode`:=11
5711   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5712 \endgroup
5713 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5714   \let\bbl@arabicjust\relax
5715   \newattribute\bblar@kashida
5716   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5717   \bblar@kashida=\z@
5718   \bbl@patchfont{\bbl@parsejalt}}%
5719   \directlua{
5720     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5721     Babel.arabic.elong_map[\the\localeid] = {}
5722     luatexbase.add_to_callback('post_linebreak_filter',
5723       Babel.arabic.justify, 'Babel.arabic.justify')
5724     luatexbase.add_to_callback('hpack_filter',
5725       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5726   }}%

```

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

```

5727 \def\bblar@fetchjalt#1#2#3#4{%
5728   \bbl@exp{\bbl@foreach{#1}}{%
5729     \bbl@ifunset\bblar@JE@##1}%
5730     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5731     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5732   \directlua{%
5733     local last = nil
5734     for item in node.traverse(tex.box[0].head) do
5735       if item.id == node.id'glyph' and item.char > 0x600 and
5736         not (item.char == 0x200D) then
5737         last = item
5738       end
5739     end
5740     Babel.arabic.#3['##1#4'] = last.char
5741   }}

```

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

```

5742 \gdef\bbl@parsejalt{%
5743   \ifx\addfontfeature\undefined\else

```

```

5744 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5745 \ifin@
5746 \directlua{%
5747     if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5748         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5749         tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5750     end
5751 }%
5752 \fi
5753 \fi}
5754 \gdef\bbl@parsejalti{%
5755 \beginingroup
5756 \let\bbl@parsejalt\relax % To avoid infinite loop
5757 \edef\bbl@tempb{\fontid\font}%
5758 \bblar@nofswarn
5759 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5760 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5761 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5762 \addfontfeature{RawFeature+=jalt}%
5763 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5764 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5765 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5766 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5767 \directlua{%
5768     for k, v in pairs(Babel.arabic.from) do
5769         if Babel.arabic.dest[k] and
5770             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5771             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5772                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5773         end
5774     end
5775 }%
5776 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5777 \beginingroup
5778 \catcode`#=11
5779 \catcode`~=11
5780 \directlua{
5781
5782 Babel.arabic = Babel.arabic or {}
5783 Babel.arabic.from = {}
5784 Babel.arabic.dest = {}
5785 Babel.arabic.justify_factor = 0.95
5786 Babel.arabic.justify_enabled = true
5787 Babel.arabic.kashida_limit = -1
5788
5789 function Babel.arabic.justify(head)
5790     if not Babel.arabic.justify_enabled then return head end
5791     for line in node.traverse_id(node.id'hlist', head) do
5792         Babel.arabic.justify_hlist(head, line)
5793     end
5794     return head
5795 end
5796
5797 function Babel.arabic.justify_hbox(head, gc, size, pack)
5798     local has_inf = false
5799     if Babel.arabic.justify_enabled and pack == 'exactly' then
5800         for n in node.traverse_id(12, head) do
5801             if n.stretch_order > 0 then has_inf = true end
5802         end
5803         if not has_inf then
5804             Babel.arabic.justify_hlist(head, nil, gc, size, pack)

```

```

5805     end
5806 end
5807 return head
5808 end
5809
5810 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5811     local d, new
5812     local k_list, k_item, pos_inline
5813     local width, width_new, full, k_curr, wt_pos, goal, shift
5814     local subst_done = false
5815     local elong_map = Babel.arabic.elong_map
5816     local cnt
5817     local last_line
5818     local GLYPH = node.id'glyph'
5819     local KASHIDA = Babel.attr_kashida
5820     local LOCALE = Babel.attr_locale
5821     local first_pass
5822
5823     if line == nil then
5824         line = {}
5825         line.glue_sign = 1
5826         line.glue_order = 0
5827         line.head = head
5828         line.shift = 0
5829         line.width = size
5830     end
5831
5832     % Exclude last line. todo. But-- it discards one-word lines, too!
5833     % ? Look for glue = 12:15
5834     if (line.glue_sign == 1 and line.glue_order == 0) then
5835         elongs = {}      % Stores elongated candidates of each line
5836         k_list = {}      % And all letters with kashida
5837         pos_inline = 0   % Not yet used
5838
5839         for n in node.traverse_id(GLYPH, line.head) do
5840             pos_inline = pos_inline + 1 % To find where it is. Not used.
5841
5842             % Elongated glyphs
5843             if elong_map then
5844                 local locale = node.get_attribute(n, LOCALE)
5845                 if elong_map[locale] and elong_map[locale][n.font] and
5846                     elong_map[locale][n.font][n.char] then
5847                     table.insert(elongs, {node = n, locale = locale} )
5848                     node.set_attribute(n.prev, KASHIDA, 0)
5849                 end
5850             end
5851
5852             % Tatwil. First create a list of nodes marked with kashida. The
5853             % rest of nodes can be ignored. The list of used weights is build
5854             % when transforms with the key kashida= are declared.
5855             if Babel.kashida_wts then
5856                 local k_wt = node.get_attribute(n, KASHIDA)
5857                 if k_wt > 0 then % todo. parameter for multi inserts
5858                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5859                 end
5860             end
5861
5862             end % of node.traverse_id
5863
5864             if #elongs == 0 and #k_list == 0 then goto next_line end
5865             full = line.width
5866             shift = line.shift
5867             goal = full * Babel.arabic.justify_factor % A bit crude

```

```

5868 width = node.dimensions(line.head)    % The 'natural' width
5869
5870 % == Elongated ==
5871 % Original idea taken from 'chickenize'
5872 while (#elongs > 0 and width < goal) do
5873     subst_done = true
5874     local x = #elongs
5875     local curr = elongs[x].node
5876     local oldchar = curr.char
5877     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5878     width = node.dimensions(line.head) % Check if the line is too wide
5879     % Substitute back if the line would be too wide and break:
5880     if width > goal then
5881         curr.char = oldchar
5882         break
5883     end
5884     % If continue, pop the just substituted node from the list:
5885     table.remove(elongs, x)
5886 end
5887
5888 % == Tatwil ==
5889 % Traverse the kashida node list so many times as required, until
5890 % the line is filled. The first pass adds a tatweel after each
5891 % node with kashida in the line, the second pass adds another one,
5892 % and so on. In each pass, add first the kashida with the highest
5893 % weight, then with lower weight and so on.
5894 if #k_list == 0 then goto next_line end
5895
5896 width = node.dimensions(line.head)    % The 'natural' width
5897 k_curr = #k_list % Traverse backwards, from the end
5898 wt_pos = 1
5899
5900 while width < goal do
5901     subst_done = true
5902     k_item = k_list[k_curr].node
5903     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5904         d = node.copy(k_item)
5905         d.char = 0x0640
5906         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5907         d.xoffset = 0
5908         line.head, new = node.insert_after(line.head, k_item, d)
5909         width_new = node.dimensions(line.head)
5910         if width > goal or width == width_new then
5911             node.remove(line.head, new) % Better compute before
5912             break
5913         end
5914         if Babel.fix_diacr then
5915             Babel.fix_diacr(k_item.next)
5916         end
5917         width = width_new
5918     end
5919     if k_curr == 1 then
5920         k_curr = #k_list
5921         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5922     else
5923         k_curr = k_curr - 1
5924     end
5925 end
5926
5927 % Limit the number of tatweel by removing them. Not very efficient,
5928 % but it does the job in a quite predictable way.
5929 if Babel.arabic.kashida_limit > -1 then
5930     cnt = 0

```

```

5931     for n in node.traverse_id(GLYPH, line.head) do
5932         if n.char == 0x0640 then
5933             cnt = cnt + 1
5934             if cnt > Babel.arabic.kashida_limit then
5935                 node.remove(line.head, n)
5936             end
5937         else
5938             cnt = 0
5939         end
5940     end
5941 end
5942
5943 ::next_line::
5944
5945 % Must take into account marks and ins, see luatex manual.
5946 % Have to be executed only if there are changes. Investigate
5947 % what's going on exactly.
5948 if subst_done and not gc then
5949     d = node.hpack(line.head, full, 'exactly')
5950     d.shift = shift
5951     node.insert_before(head, line, d)
5952     node.remove(head, line)
5953 end
5954 end % if process line
5955 end
5956 }
5957 \endgroup
5958 \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`.

```

5959 \def\bbl@scr@node@list{%
5960   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
5961   ,Greek,Latin,Old Church Slavonic Cyrillic,}
5962 \ifnum\bbl@bidimode=102 % bidi-r
5963   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
5964 \fi
5965 \def\bbl@set@renderer{%
5966   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
5967   \ifin@
5968     \let\bbl@unset@renderer\relax
5969   \else
5970     \bbl@exp{%
5971       \def\\bbl@unset@renderer{%
5972         \def<g__fontspec_default_fontopts_clist>{%
5973           \[g__fontspec_default_fontopts_clist]}}%
5974       \def<g__fontspec_default_fontopts_clist>{%
5975         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}}%
5976     \fi}
5977 <@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.

```

5978% TODO - to a lua file
5979 \directlua{% DL6
5980 Babel.script_blocks = {
5981   ['dflt'] = {},
5982   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5983               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5984   ['Armn'] = {{0x0530, 0x058F}},
5985   ['Beng'] = {{0x0980, 0x09FF}},
5986   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0ABBF}},
5987   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5988   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5989               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5990   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5991   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5992               {0xAB00, 0xAB2F}},
5993   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5994   % Don't follow strictly Unicode, which places some Coptic letters in
5995   % the 'Greek and Coptic' block
5996   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5997   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5998               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5999               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6000               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6001               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6002               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6003   ['Hebr'] = {{0x0590, 0x05FF}},
6004   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6005               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6006   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6007   ['Knda'] = {{0x0C80, 0x0CFF}},
6008   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6009               {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6010               {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6011   ['Lao'] = {{0x0E80, 0x0EFF}},
6012   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6013               {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6014               {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6015   ['Mahj'] = {{0x11150, 0x1117F}},
6016   ['Mlym'] = {{0x0D00, 0x0D7F}},
6017   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6018   ['Orya'] = {{0x0B00, 0x0B7F}},
6019   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6020   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6021   ['Taml'] = {{0x0B80, 0x0BFF}},
6022   ['Telu'] = {{0x0C00, 0x0C7F}},
6023   ['Tfng'] = {{0x2D30, 0x2D7F}},
6024   ['Thai'] = {{0x0E00, 0x0E7F}},
6025   ['Tibt'] = {{0x0F00, 0x0FFF}},
6026   ['Vaii'] = {{0xA500, 0xA63F}},
6027   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6028 }
6029
6030 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6031 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6032 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6033
6034 function Babel.locale_map(head)
6035   if not Babel.locale_mapped then return head end
6036
6037   local LOCALE = Babel.attr_locale

```

```

6038 local GLYPH = node.id('glyph')
6039 local inmath = false
6040 local toloc_save
6041 for item in node.traverse(head) do
6042   local toloc
6043   if not inmath and item.id == GLYPH then
6044     % Optimization: build a table with the chars found
6045     if Babel.chr_to_loc[item.char] then
6046       toloc = Babel.chr_to_loc[item.char]
6047     else
6048       for lc, maps in pairs(Babel.loc_to_scr) do
6049         for _, rg in pairs(maps) do
6050           if item.char >= rg[1] and item.char <= rg[2] then
6051             Babel.chr_to_loc[item.char] = lc
6052             toloc = lc
6053             break
6054           end
6055         end
6056       end
6057       % Treat composite chars in a different fashion, because they
6058       % 'inherit' the previous locale.
6059       if (item.char >= 0x0300 and item.char <= 0x036F) or
6060         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6061         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6062         Babel.chr_to_loc[item.char] = -2000
6063         toloc = -2000
6064       end
6065       if not toloc then
6066         Babel.chr_to_loc[item.char] = -1000
6067       end
6068     end
6069     if toloc == -2000 then
6070       toloc = toloc_save
6071     elseif toloc == -1000 then
6072       toloc = nil
6073     end
6074     if toloc and Babel.locale_props[toloc] and
6075       Babel.locale_props[toloc].letters and
6076       tex.getcatcode(item.char) \string~= 11 then
6077       toloc = nil
6078     end
6079     if toloc and Babel.locale_props[toloc].script
6080       and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6081       and Babel.locale_props[toloc].script ==
6082       Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6083       toloc = nil
6084     end
6085     if toloc then
6086       if Babel.locale_props[toloc].lg then
6087         item.lang = Babel.locale_props[toloc].lg
6088         node.set_attribute(item, LOCALE, toloc)
6089       end
6090       if Babel.locale_props[toloc]['/'..item.font] then
6091         item.font = Babel.locale_props[toloc]['/'..item.font]
6092       end
6093     end
6094     toloc_save = toloc
6095   elseif not inmath and item.id == 7 then % Apply recursively
6096     item.replace = item.replace and Babel.locale_map(item.replace)
6097     item.pre      = item.pre and Babel.locale_map(item.pre)
6098     item.post     = item.post and Babel.locale_map(item.post)
6099   elseif item.id == node.id'math' then
6100     inmath = (item.subtype == 0)

```



```

6101   end
6102 end
6103 return head
6104 end
6105 }

```

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

```

6106 \newcommand\babelcharproperty[1]{%
6107   \count@=#1\relax
6108   \ifvmode
6109     \expandafter\babel@chprop
6110   \else
6111     \babel@error{charproperty-only-vertical}{}{}{}%
6112   \fi}
6113 \newcommand\babel@chprop[3][\the\count@]{%
6114   \@tempcnta=#1\relax
6115   \babel@ifunset{babel@chprop@#2}% {unknown-char-property}
6116     {\babel@error{unknown-char-property}{}{#2}{}%
6117     }%
6118   \loop
6119     \babel@cs{chprop@#2}{#3}%
6120   \ifnum\count@<\@tempcnta
6121     \advance\count@\@ne
6122   \repeat}
6123 \def\babel@chprop@direction#1{%
6124   \directlua{
6125     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6126     Babel.characters[\the\count@]['d'] = '#1'
6127   }}
6128 \let\babel@chprop@bc\babel@chprop@direction
6129 \def\babel@chprop@mirror#1{%
6130   \directlua{
6131     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6132     Babel.characters[\the\count@]['m'] = '\number#1'
6133   }}
6134 \let\babel@chprop@bmg\babel@chprop@mirror
6135 \def\babel@chprop@linebreak#1{%
6136   \directlua{
6137     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6138     Babel.cjk_characters[\the\count@]['c'] = '#1'
6139   }}
6140 \let\babel@chprop@lb\babel@chprop@linebreak
6141 \def\babel@chprop@locale#1{%
6142   \directlua{
6143     Babel.chr_to_loc = Babel.chr_to_loc or {}
6144     Babel.chr_to_loc[\the\count@] =
6145       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@@#1}}\space
6146   }}

```

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.

```

6147 \directlua{% DL7
6148   Babel.nohyphenation = \the\l@nohyphenation
6149 }

```

Now the \TeX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the `{n}` syntax. For example, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect is not dissimilar to lua `load` – save the code as string in a \TeX macro, and expand this macro at the

appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```

6150 \beginingroup
6151 \catcode\~ = 12
6152 \catcode\% = 12
6153 \catcode\& = 14
6154 \catcode\| = 12
6155 \gdef\babelprehyphenation{%&
6156   \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{}}
6157 \gdef\babelposthyphenation{%&
6158   \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}{}}
6159 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6160   \ifcase#1
6161     \bbl@activateprehyphen
6162   \or
6163     \bbl@activateposthyphen
6164   \fi
6165 \beginingroup
6166   \def\babeltempa{\bbl@add@list\babeltempb}%&
6167   \let\babeltempb\empty
6168   \def\bbl@tempa{#5}%&
6169   \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6170   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6171     \bbl@ifsamestring{##1}{remove}%&
6172     {\bbl@add@list\babeltempb{nil}}}%&
6173     {\directlua{
6174       local rep = {[##1]=}
6175       local three_args = '%s*=%s*([%- %d%.%a{}]|+)%s+([%- %d%.%a{}]|+)%s+([%- %d%.%a{}]|+)'
6176       & Numeric passes directly: kern, penalty...
6177       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6178       rep = rep:gsub('^%s*(insert)%s*', ', 'insert = true, ')
6179       rep = rep:gsub('^%s*(after)%s*', ', 'after = true, ')
6180       rep = rep:gsub('(string)%s*=%s*([%^s,]*)', Babel.capture_func)
6181       rep = rep:gsub('node%s*=%s*(%a+)%s*(%a*)', Babel.capture_node)
6182       rep = rep:gsub(' (norule)' .. three_args,
6183         'norule = {' .. '%2, %3, %4' .. '}')
6184       if #1 == 0 or #1 == 2 then
6185         rep = rep:gsub(' (space)' .. three_args,
6186           'space = {' .. '%2, %3, %4' .. '}')
6187         rep = rep:gsub(' (spacefactor)' .. three_args,
6188           'spacefactor = {' .. '%2, %3, %4' .. '}')
6189         rep = rep:gsub('(kashida)%s*=%s*([%^s,]*)', Babel.capture_kashida)
6190         & Transform values
6191         rep, n = rep:gsub(' ({[%a%-%.]+)|([%a%-_.]+)}',
6192           function(v,d)
6193             return string.format (
6194               '{\the\curname bbl@id@@#3\endcsname,"%s",%s}',
6195               v,
6196               load( 'return Babel.locale_props'..
6197                 '[\the\curname bbl@id@@#3\endcsname].' .. d)() )
6198             end )
6199         rep, n = rep:gsub(' ({[%a%-%.]+)|([%- %d%.]+)}',
6200           '{\the\curname bbl@id@@#3\endcsname,"%1",%2}')
6201       end
6202       if #1 == 1 then
6203         rep = rep:gsub(' (no)%s*=%s*([%^s,]*)', Babel.capture_func)
6204         rep = rep:gsub(' (pre)%s*=%s*([%^s,]*)', Babel.capture_func)
6205         rep = rep:gsub(' (post)%s*=%s*([%^s,]*)', Babel.capture_func)
6206       end
6207       tex.print([[\string\babeltempa{[]] .. rep .. [[]]])
6208     }]}%&
6209 \bbl@foreach\babeltempb{%&
6210   \bbl@forky{##1}{%&

```

```

6211 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6212 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6213 \ifin@else
6214 \bbl@error{bad-transform-option}{###1}{}}&%
6215 \fi}}&%
6216 \let\bbl@kv@attribute\relax
6217 \let\bbl@kv@label\relax
6218 \let\bbl@kv@fonts\@empty
6219 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6220 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6221 \ifx\bbl@kv@attribute\relax
6222 \ifx\bbl@kv@label\relax\else
6223 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6224 \bbl@replace\bbl@kv@fonts{ },}&%
6225 \edef\bbl@kv@attribute{\bbl@ATR@bbl@kv@label @#3@bbl@kv@fonts}&%
6226 \count@z@
6227 \def\bbl@elt##1##2##3{&%
6228 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6229 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6230 {\count@z@ne}&%
6231 {\bbl@error{font-conflict-transforms}{}}{}}&%
6232 {}}&%
6233 \bbl@transfont@list
6234 \ifnum\count@z@
6235 \bbl@exp{\global\bbl@add\bbl@transfont@list
6236 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6237 \fi
6238 \bbl@ifunset{\bbl@kv@attribute}&%
6239 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6240 {}}&%
6241 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6242 \fi
6243 \else
6244 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6245 \fi
6246 \directlua{
6247 local lbkr = Babel.linebreaking.replacements[#1]
6248 local u = unicode.utf8
6249 local id, attr, label
6250 if #1 == 0 then
6251 id = \the\csname bbl@id@@#3\endcsname\space
6252 else
6253 id = \the\csname l@#3\endcsname\space
6254 end
6255 \ifx\bbl@kv@attribute\relax
6256 attr = -1
6257 \else
6258 attr = luatexbase.registernumber'\bbl@kv@attribute'
6259 \fi
6260 \ifx\bbl@kv@label\relax\else &% Same refs:
6261 label = [==[\bbl@kv@label]==]
6262 \fi
6263 &% Convert pattern:
6264 local patt = string.gsub([==[#4]==], '%s', '')
6265 if #1 == 0 then
6266 patt = string.gsub(patt, '|', ' ')
6267 end
6268 if not u.find(patt, '()', nil, true) then
6269 patt = '()' .. patt .. '()'
6270 end
6271 if #1 == 1 then
6272 patt = string.gsub(patt, '%(%)^', '^()')
6273 patt = string.gsub(patt, '%$(%)', '()$')

```

```

6274     end
6275     patt = u.gsub(patt, '{(.)}',
6276         function (n)
6277             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6278         end)
6279     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6280         function (n)
6281             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6282         end)
6283     lbkr[id] = lbkr[id] or {}
6284     table.insert(lbkr[id],
6285         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6286 }&%
6287 \endgroup}
6288 \endgroup
6289 \let\bbl@transfont@list\@empty
6290 \def\bbl@settransfont{%
6291     \global\let\bbl@settransfont\relax % Execute only once
6292     \gdef\bbl@transfont{%
6293         \def\bbl@elt####1####2####3{%
6294             \bbl@ifblank{####3}%
6295             {\count@tw@}% Do nothing if no fonts
6296             {\count@z@
6297             \bbl@vforeach{####3}{%
6298                 \def\bbl@tempd{#####1}%
6299                 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6300                 \ifx\bbl@tempd\bbl@tempe
6301                     \count@\@ne
6302                 \else\ifx\bbl@tempd\bbl@transfam
6303                     \count@\@ne
6304                 \fi\fi}%
6305             \ifcase\count@
6306             \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6307             \or
6308             \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6309             \fi}}%
6310         \bbl@transfont@list}%
6311     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6312     \gdef\bbl@transfam{-unknown-}%
6313     \bbl@foreach\bbl@font@fams{%
6314         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6315         \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6316         {\xdef\bbl@transfam{##1}}%
6317     }}}}
6318 \DeclareRobustCommand\enablelocaletransform[1]{%
6319     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6320     {\bbl@error{transform-not-available}{#1}{}}}%
6321     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6322 \DeclareRobustCommand\disablelocaletransform[1]{%
6323     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6324     {\bbl@error{transform-not-available-b}{#1}{}}}%
6325     {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}%
6326 \def\bbl@activateposthyphen{%
6327     \let\bbl@activateposthyphen\relax
6328     \ifx\bbl@attr@hboxed\undefined
6329         \newattribute\bbl@attr@hboxed
6330     \fi
6331     \directlua{
6332         require('babel-transforms.lua')
6333         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6334     }}
6335 \def\bbl@activateprehyphen{%
6336     \let\bbl@activateprehyphen\relax

```

```

6337 \ifx\bbl@attr@hboxed\undefined
6338 \newattribute\bbl@attr@hboxed
6339 \fi
6340 \directlua{
6341   require('babel-transforms.lua')
6342   Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6343 }
6344 \newcommand\SetTransformValue[3]{%
6345 \directlua{
6346   Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6347 }}

```

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.

```

6348 \newcommand\localeprehyphenation[1]{%
6349 \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.

```

6350 \def\bbl@activate@preotf{%
6351 \let\bbl@activate@preotf\relax % only once
6352 \directlua{
6353   function Babel.pre_otfload_v(head)
6354     if Babel.numbers and Babel.digits_mapped then
6355       head = Babel.numbers(head)
6356     end
6357     if Babel.bidi_enabled then
6358       head = Babel.bidi(head, false, dir)
6359     end
6360     return head
6361   end
6362   %
6363   function Babel.pre_otfload_h(head, gc, sz, pt, dir) %%% TODO
6364     if Babel.numbers and Babel.digits_mapped then
6365       head = Babel.numbers(head)
6366     end
6367     if Babel.bidi_enabled then
6368       head = Babel.bidi(head, false, dir)
6369     end
6370     return head
6371   end
6372   %
6373   luatexbase.add_to_callback('pre_linebreak_filter',
6374     Babel.pre_otfload_v,
6375     'Babel.pre_otfload_v',
6376     luatexbase.priority_in_callback('pre_linebreak_filter',
6377       'luaotfload.node_processor') or nil)
6378   %
6379   luatexbase.add_to_callback('hpack_filter',
6380     Babel.pre_otfload_h,
6381     'Babel.pre_otfload_h',
6382     luatexbase.priority_in_callback('hpack_filter',
6383       'luaotfload.node_processor') or nil)
6384 }}

```

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.

```

6385 \breakafterdirmode=1
6386 \ifnum\bbbl@bidimode>\@ne % Any bidi= except default (=1)
6387 \let\bbbl@beforeforeign\leavevmode
6388 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6389 \RequirePackage{luatexbase}
6390 \bbbl@activate@preotf
6391 \directlua{
6392   require('babel-data-bidi.lua')
6393   \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
6394     require('babel-bidi-basic.lua')
6395   \or
6396     require('babel-bidi-basic-r.lua')
6397     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6398     table.insert(Babel.ranges, {0xF0000, 0xFFFFD, 'on'})
6399     table.insert(Babel.ranges, {0x100000, 0x10FFFD, 'on'})
6400   \fi}
6401 \newattribute\bbbl@attr@dir
6402 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
6403 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
6404 \fi
6405 \chardef\bbbl@thetextdir\z@
6406 \chardef\bbbl@thepardir\z@
6407 \def\bbbl@getluadir#1{%
6408   \directlua{
6409     if tex.#ldir == 'TLT' then
6410       tex.sprint('0')
6411     elseif tex.#ldir == 'TRT' then
6412       tex.sprint('1')
6413     else
6414       tex.sprint('0')
6415     end}}
6416 \def\bbbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6417   \ifcase#3\relax
6418     \ifcase\bbbl@getluadir{#1}\relax\else
6419       #2 TLT\relax
6420     \fi
6421   \else
6422     \ifcase\bbbl@getluadir{#1}\relax
6423       #2 TRT\relax
6424     \fi
6425   \fi}
6426 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6427 \def\bbbl@thedir{0}
6428 \def\bbbl@textdir#1{%
6429   \bbbl@setluadir{text}\textdir{#1}%
6430   \chardef\bbbl@thetextdir#1\relax
6431   \edef\bbbl@thedir{\the\numexpr\bbbl@thepardir*4+#1}%
6432   \setattribute\bbbl@attr@dir{\numexpr\bbbl@thepardir*4+#1}}
6433 \def\bbbl@pardir#1{% Used twice
6434   \bbbl@setluadir{par}\pardir{#1}%
6435   \chardef\bbbl@thepardir#1\relax}
6436 \def\bbbl@bodydir{\bbbl@setluadir{body}\bodydir}% Used once
6437 \def\bbbl@pagedir{\bbbl@setluadir{page}\pagedir}% Unused
6438 \def\bbbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6439 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6440 \def\bbbl@insidemath{0}%
6441 \def\bbbl@everymath{\def\bbbl@insidemath{1}}
6442 \def\bbbl@everydisplay{\def\bbbl@insidemath{2}}

```

```

6443 \frozen@everymath\expandafter{%
6444   \expandafter\bbl@everymath\the\frozen@everymath}
6445 \frozen@everydisplay\expandafter{%
6446   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6447 \AtBeginDocument{
6448   \directlua{
6449     function Babel.math_box_dir(head)
6450       if not (token.get_macro('bbl@insidemath') == '0') then
6451         if Babel.hlist_has_bidi(head) then
6452           local d = node.new(node.id'dir')
6453           d.dir = '+TRT'
6454           node.insert_before(head, node.has_glyph(head), d)
6455           local inmath = false
6456           for item in node.traverse(head) do
6457             if item.id == 11 then
6458               inmath = (item.subtype == 0)
6459             elseif not inmath then
6460               node.set_attribute(item,
6461                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6462             end
6463           end
6464         end
6465       end
6466       return head
6467     end
6468     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6469       "Babel.math_box_dir", 0)
6470     if Babel.unset_atdir then
6471       luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6472         "Babel.unset_atdir")
6473       luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6474         "Babel.unset_atdir")
6475     end
6476   }}%
6477 \fi

Experimental. Tentative name.

6478 \DeclareRobustCommand\localebox[1]{%
6479   {\def\bbl@insidemath{0}%
6480     \mbox{\foreignlanguage{\language}{#1}}}}

```

10.12 Layout

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

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

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

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

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

```

6481 \bbl@trace{Redefinitions for bidi layout}

```

```

6482 %
6483 <<{*More package options}>> ≡
6484 \chardef\bbl@eqnpos\z@
6485 \DeclareOption{leqno}{\chardef\bbl@eqnpos@ne}
6486 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6487 <</More package options>>
6488 %
6489 \ifnum\bbl@bidimode>\z@ % Any bidi=
6490 \matheqdirmode@ne % A luatex primitive
6491 \let\bbl@eqnodir\relax
6492 \def\bbl@eqdel{()}
6493 \def\bbl@eqnum{%
6494   {\normalfont\normalcolor
6495     \expandafter\@firstoftwo\bbl@eqdel
6496     \theequation
6497     \expandafter\@secondoftwo\bbl@eqdel}}
6498 \def\bbl@puteqno#1{\eqno\hbox{#1}}
6499 \def\bbl@putleqno#1{\leqno\hbox{#1}}
6500 \def\bbl@eqno@flip#1{%
6501   \ifdim\predisplaysize=-\maxdimen
6502     \eqno
6503     \hb@xt@.01pt{%
6504       \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6505   \else
6506     \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6507   \fi
6508   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6509 \def\bbl@leqno@flip#1{%
6510   \ifdim\predisplaysize=-\maxdimen
6511     \leqno
6512     \hb@xt@.01pt{%
6513       \hss\hb@xt@\displaywidth{{#1}\glet\bbl@upset\@currentlabel}\hss}}%
6514   \else
6515     \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6516   \fi
6517   \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6518 \AtBeginDocument{%
6519   \ifx\bbl@noamsmath\relax\else
6520   \ifx\maketag@@@undefined % Normal equation, eqnarray
6521     \AddToHook{env/equation/begin}{%
6522       \ifnum\bbl@thetextdir>\z@
6523         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6524         \let\@eqnnum\bbl@eqnum
6525         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6526         \chardef\bbl@thetextdir\z@
6527         \bbl@add\normalfont{\bbl@eqnodir}%
6528         \ifcase\bbl@eqnpos
6529           \let\bbl@puteqno\bbl@eqno@flip
6530         \or
6531           \let\bbl@puteqno\bbl@leqno@flip
6532         \fi
6533       \fi}%
6534   \ifnum\bbl@eqnpos=\tw@\else
6535     \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6536   \fi
6537   \AddToHook{env/eqnarray/begin}{%
6538     \ifnum\bbl@thetextdir>\z@
6539       \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6540       \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6541       \chardef\bbl@thetextdir\z@
6542       \bbl@add\normalfont{\bbl@eqnodir}%
6543     \ifnum\bbl@eqnpos=@ne
6544       \def\@eqnnum{%

```



```

6545         \setbox\z@\hbox{\bbl@eqnum}%
6546         \hbox to 0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6547     \else
6548         \let\@eqnnum\bbl@eqnum
6549     \fi
6550 \fi}
6551 % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6552 \expandafter\bbl@sreplace\csname\ \endcsname{$$}{\eqno\kern.001pt$}$%
6553 \else % amstex
6554     \bbl@exp{% Hack to hide maybe undefined conditionals:
6555         \chardef\bbl@eqnpos=0%
6556         \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6557     \ifnum\bbl@eqnpos=\@ne
6558         \let\bbl@ams@lap\hbox
6559     \else
6560         \let\bbl@ams@lap\llap
6561     \fi
6562     \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6563     \bbl@sreplace\intertext@\{\normalbaselines}%
6564     {\normalbaselines
6565       \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6566     \ExplSyntaxOff
6567     \def\bbl@ams@tagbox#1#2{\#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6568     \ifx\bbl@ams@lap\hbox % leqno
6569         \def\bbl@ams@flip#1{%
6570             \hbox to 0.01pt{\hss\hbox to\displaywidth{\#1}\hss}}}%
6571     \else % eqno
6572         \def\bbl@ams@flip#1{%
6573             \hbox to 0.01pt{\hbox to\displaywidth{\hss{\#1}\hss}}}%
6574     \fi
6575     \def\bbl@ams@preset#1{%
6576         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6577         \ifnum\bbl@thetextdir>\z@
6578             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6579             \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6580             \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6581         \fi}%
6582     \ifnum\bbl@eqnpos=\tw@ \else
6583         \def\bbl@ams@equation{%
6584             \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6585             \ifnum\bbl@thetextdir>\z@
6586                 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6587                 \chardef\bbl@thetextdir\z@
6588                 \bbl@add\normalfont{\bbl@eqnodir}%
6589                 \ifcase\bbl@eqnpos
6590                     \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6591                 \or
6592                     \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6593                 \fi
6594             \fi}%
6595     \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6596     \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6597 \fi
6598 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6599 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6600 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6601 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6602 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6603 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6604 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6605 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6606 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6607 % Hackish, for proper alignment. Don't ask me why it works!:
```

```

6608 \bbl@exp{% Avoid a 'visible' conditional
6609 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}%
6610 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6611 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6612 \AddToHook{env/split/before}{%
6613 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6614 \ifnum\bbl@thetextdir>\z@
6615 \bbl@ifsamestring\@currentenv{equation}%
6616 {\ifx\bbl@ams@lap\hbox % leqno
6617 \def\bbl@ams@flip#1{%
6618 \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6619 \else
6620 \def\bbl@ams@flip#1{%
6621 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#{1}}}}%
6622 \fi}%
6623 }%
6624 \fi}%
6625 \fi\fi}
6626 \fi
6627 \def\bbl@provide@extra#1{%
6628 % == onchar ==
6629 \ifx\bbl@KVP@onchar\@nnil\else
6630 \bbl@luahyphenate
6631 \bbl@exp{%
6632 \\\AddToHook{env/document/before}{\select@language{#1}}}%
6633 \directlua{
6634 if Babel.locale_mapped == nil then
6635 Babel.locale_mapped = true
6636 Babel.linebreaking.add_before(Babel.locale_map, 1)
6637 Babel.loc_to_scr = {}
6638 Babel.chr_to_loc = Babel.chr_to_loc or {}
6639 end
6640 Babel.locale_props[\the\localeid].letters = false
6641 }%
6642 \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6643 \ifin@
6644 \directlua{
6645 Babel.locale_props[\the\localeid].letters = true
6646 }%
6647 \fi
6648 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6649 \ifin@
6650 \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6651 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6652 \fi
6653 \bbl@exp{\\bbl@add\\bbl@starthyphens
6654 {\\bbl@patterns@lua{\language\language}}}%
6655 %^A add error/warning if no script
6656 \directlua{
6657 if Babel.script_blocks['\bbl@cl{sbc}'] then
6658 Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6659 Babel.locale_props[\the\localeid].lg = \the\@nameuse{l\language}\space
6660 end
6661 }%
6662 \fi
6663 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6664 \ifin@
6665 \bbl@ifunset{bbl@lsys\language}{\bbl@provide@lsys\language}%
6666 \bbl@ifunset{bbl@wdir\language}{\bbl@provide@dirs\language}%
6667 \directlua{
6668 if Babel.script_blocks['\bbl@cl{sbc}'] then
6669 Babel.loc_to_scr[\the\localeid] =
6670 Babel.script_blocks['\bbl@cl{sbc}']

```

```

6671     end}%
6672     \ifx\bbbl@mapselect\@undefined % TODO. almost the same as mapfont
6673     \AtBeginDocument{%
6674         \bbbl@patchfont{\bbbl@mapselect}}%
6675         {\selectfont}}%
6676     \def\bbbl@mapselect{%
6677         \let\bbbl@mapselect\relax
6678         \edef\bbbl@prefontid{\fontid\font}}%
6679     \def\bbbl@mapdir##1{%
6680         \begingroup
6681         \setbox\z@\hbox{% Force text mode
6682             \def\language{##1}%
6683             \let\bbbl@ifrestoring\@firstoftwo % To avoid font warning
6684             \bbbl@switchfont
6685             \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6686                 \directlua{
6687                     Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6688                     ['/\bbbl@prefontid'] = \fontid\font\sname}%
6689                 \fi}%
6690         \endgroup}%
6691     \fi
6692     \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
6693     \fi
6694     % TODO - catch non-valid values
6695 \fi
6696 % == mapfont ==
6697 % For bidi texts, to switch the font based on direction. Old.
6698 \ifx\bbbl@KVP@mapfont\@nnil\else
6699     \bbbl@ifsamestring{\bbbl@KVP@mapfont}{direction}}{%
6700         {\bbbl@error{unknown-mapfont}}{}}}%
6701     \bbbl@ifunset{\bbbl@lsys\language}{\bbbl@provide@lsys{\language}}}%
6702     \bbbl@ifunset{\bbbl@wdir\language}{\bbbl@provide@dirs{\language}}}%
6703     \ifx\bbbl@mapselect\@undefined % TODO. See onchar.
6704         \AtBeginDocument{%
6705             \bbbl@patchfont{\bbbl@mapselect}}%
6706             {\selectfont}}%
6707         \def\bbbl@mapselect{%
6708             \let\bbbl@mapselect\relax
6709             \edef\bbbl@prefontid{\fontid\font}}%
6710     \def\bbbl@mapdir##1{%
6711         {\def\language{##1}%
6712             \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
6713             \bbbl@switchfont
6714             \directlua{Babel.fontmap
6715                 [\the\csname bbl@wdir@##1\endcsname]%
6716                 [\bbbl@prefontid]=\fontid\font}}}%
6717     \fi
6718     \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\language}}}%
6719 \fi
6720 % == Line breaking: CJK quotes ==
6721 \ifcase\bbbl@engine\or
6722     \bbbl@xin{/c}{/\bbbl@cl{\lnbrk}}%
6723 \ifin@
6724     \bbbl@ifunset{\bbbl@quote\language}}{%
6725         {\directlua{
6726             Babel.locale_props[\the\localeid].cjk_quotes = {}
6727             local cs = 'op'
6728             for c in string.utfvalues(
6729                 [[\csname bbl@quote\language\endcsname]]) do
6730                 if Babel.cjk_characters[c].c == 'qu' then
6731                     Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6732                 end
6733                 cs = ( cs == 'op') and 'cl' or 'op'

```

```

6734         end
6735     } }%
6736 \fi
6737 \fi
6738 % == Counters: mapdigits ==
6739 % Native digits
6740 \ifx\bbbl@KVP@mapdigits\@nnil\else
6741     \bbbl@ifunset{bbbl@dgnat@\languagename}{ }%
6742     {\RequirePackage{luatexbase}%
6743     \bbbl@activate@preotf
6744     \directlua{
6745         Babel.digits_mapped = true
6746         Babel.digits = Babel.digits or {}
6747         Babel.digits[\the\localeid] =
6748             table.pack(string.utfvalue('\bbbl@cldgnat'))
6749         if not Babel.numbers then
6750             function Babel.numbers(head)
6751                 local LOCALE = Babel.attr_locale
6752                 local GLYPH = node.id'glyph'
6753                 local inmath = false
6754                 for item in node.traverse(head) do
6755                     if not inmath and item.id == GLYPH then
6756                         local temp = node.get_attribute(item, LOCALE)
6757                         if Babel.digits[temp] then
6758                             local chr = item.char
6759                             if chr > 47 and chr < 58 then
6760                                 item.char = Babel.digits[temp][chr-47]
6761                             end
6762                         end
6763                         elseif item.id == node.id'math' then
6764                             inmath = (item.subtype == 0)
6765                         end
6766                     end
6767                 return head
6768             end
6769         end
6770     } }%
6771 \fi
6772 % == transforms ==
6773 \ifx\bbbl@KVP@transforms\@nnil\else
6774     \def\bbbl@elt##1##2##3{%
6775         \in@{ $transforms. } { ##1 }%
6776         \ifin@
6777             \def\bbbl@tempa{ ##1 }%
6778             \bbbl@replace\bbbl@tempa{ transforms. } { }%
6779             \bbbl@carg\bbbl@transforms{ babel\bbbl@tempa } { ##2 } { ##3 }%
6780         \fi}%
6781 \bbbl@exp{%
6782     \\bbbl@ifblank{ \bbbl@cldgnat }%
6783     { \let \\bbbl@tempa\relax }%
6784     { \def \\bbbl@tempa{%
6785         \\bbbl@elt{ transforms.prehyphenation }%
6786         { digits.native.1.0 } { ([0-9]) }%
6787         \\bbbl@elt{ transforms.prehyphenation }%
6788         { digits.native.1.1 } { string={1\string|0123456789\string|\bbbl@cldgnat} } } }%
6789 \ifx\bbbl@tempa\relax\else
6790     \toks@ \expandafter\expandafter\expandafter{%
6791         \csname bbl@inidata@\languagename\endcsname }%
6792     \bbbl@csarg\edef{ inidata@\languagename }{%
6793         \unexpanded\expandafter{ \bbbl@tempa }%
6794         \the\toks@ }%
6795 \fi
6796 \csname bbl@inidata@\languagename\endcsname

```

```

6797 \bbl@release@transforms\relax % \relax closes the last item.
6798 \fi}

Start tabular here:

6799 \def\localerestoredirs{%
6800 \ifcase\bbl@thetextdir
6801 \ifnum\textdirection=\z@\else\textdir TLT\fi
6802 \else
6803 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6804 \fi
6805 \ifcase\bbl@thepardir
6806 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6807 \else
6808 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6809 \fi}
6810 \IfBabelLayout{tabular}%
6811 {\chardef\bbl@tabular@mode\tw@}% All RTL
6812 {\IfBabelLayout{notabular}%
6813 {\chardef\bbl@tabular@mode\z@}%
6814 {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6815 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6816 % Redefine: vrules mess up dirs. TODO: why?
6817 \def\@arstrut{\relax\copy\@arstrutbox}%
6818 \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6819 \let\bbl@parabefore\relax
6820 \AddToHook{para/before}{\bbl@parabefore}
6821 \AtBeginDocument{%
6822 \bbl@replace\@tabular{$}{$%
6823 \def\bbl@insidemath{0}%
6824 \def\bbl@parabefore{\localerestoredirs}}%
6825 \ifnum\bbl@tabular@mode=\@ne
6826 \bbl@ifunset{@tabclassz}{}{%
6827 \bbl@exp{% Hide conditionals
6828 \\\bbl@sreplace\\\@tabclassz
6829 {\<ifcase>\\\@chnum}%
6830 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6831 \@ifpackageloaded{colortbl}%
6832 {\bbl@sreplace\@classz
6833 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6834 {\@ifpackageloaded{array}%
6835 {\bbl@exp{% Hide conditionals
6836 \\\bbl@sreplace\\\@classz
6837 {\<ifcase>\\\@chnum}%
6838 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6839 \\\bbl@sreplace\\\@classz
6840 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6841 {}}}%
6842 \fi}%
6843 \or % 2 = All RTL - tabular
6844 \let\bbl@parabefore\relax
6845 \AddToHook{para/before}{\bbl@parabefore}%
6846 \AtBeginDocument{%
6847 \@ifpackageloaded{colortbl}%
6848 {\bbl@replace\@tabular{$}{$%
6849 \def\bbl@insidemath{0}%
6850 \def\bbl@parabefore{\localerestoredirs}}%
6851 \bbl@sreplace\@classz
6852 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}}%
6853 {}}}%
6854 \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.

```

6855 \AtBeginDocument{%
6856   \@ifpackageloaded{multicol}%
6857     {\toks\expandafter{\multi@column@out}%
6858       \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6859     {}%
6860   \@ifpackageloaded{paracol}%
6861     {\edef\pcol@output{%
6862       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6863     {}}%
6864 \fi
6865 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

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

```

6866 \ifnum\bbl@bidimode>\z@ % Any bidi=
6867   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6868     \bbl@exp{%
6869       \mathdir\the\bodydir
6870       #1%           Once entered in math, set boxes to restore values
6871       \def\\bbl@insidemath{0}%
6872       \<ifmmode>%
6873         \everyvbox{%
6874           \the\everyvbox
6875           \bodydir\the\bodydir
6876           \mathdir\the\mathdir
6877           \everyhbox{\the\everyhbox}%
6878           \everyvbox{\the\everyvbox}}%
6879         \everyhbox{%
6880           \the\everyhbox
6881           \bodydir\the\bodydir
6882           \mathdir\the\mathdir
6883           \everyhbox{\the\everyhbox}%
6884           \everyvbox{\the\everyvbox}}%
6885         \<fi>}}%
6886   \def\@hangfrom#1{%
6887     \setbox\@tempboxa\hbox{#1}%
6888     \hangindent\wd\@tempboxa
6889     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6890       \shapemode\@ne
6891     \fi
6892     \noindent\box\@tempboxa}
6893 \fi
6894 \IfBabelLayout{tabular}
6895   {\let\bbl@OL@tabular\@tabular
6896     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6897     \let\bbl@NL@tabular\@tabular
6898     \AtBeginDocument{%
6899       \ifx\bbl@NL@tabular\@tabular\else
6900         \bbl@exp{\in@{\bbl@nextfake}{\@tabular}}}%
6901       \ifin\else
6902         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6903       \fi
6904       \let\bbl@NL@tabular\@tabular
6905     \fi}}
6906 {}
6907 \IfBabelLayout{lists}
6908   {\let\bbl@OL@list\list
6909     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6910     \let\bbl@NL@list\list
6911     \def\bbl@listparshape#1#2#3{%
6912       \parshape #1 #2 #3 %

```

```

6913 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6914 \shapemode\tw@
6915 \fi}}
6916 {}
6917 \IfBabelLayout{graphics}
6918 {\let\bbl@pictresetdir\relax
6919 \def\bbl@pictsetdir#1{%
6920 \ifcase\bbl@thetextdir
6921 \let\bbl@pictresetdir\relax
6922 \else
6923 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6924 \or\textdir TLT
6925 \else\bodydir TLT \textdir TLT
6926 \fi
6927 % \{(text|par)dir required in pgf:
6928 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6929 \fi}%
6930 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6931 \directlua{
6932 Babel.get_picture_dir = true
6933 Babel.picture_has_bidi = 0
6934 %
6935 function Babel.picture_dir (head)
6936 if not Babel.get_picture_dir then return head end
6937 if Babel.hlist_has_bidi(head) then
6938 Babel.picture_has_bidi = 1
6939 end
6940 return head
6941 end
6942 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6943 "Babel.picture_dir")
6944 }%
6945 \AtBeginDocument{%
6946 \def\LS@rot{%
6947 \setbox\@outputbox\vbox{%
6948 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6949 \long\def\put(#1,#2)#3{%
6950 \@killglue
6951 % Try:
6952 \ifx\bbl@pictresetdir\relax
6953 \def\bbl@tempc{0}%
6954 \else
6955 \directlua{
6956 Babel.get_picture_dir = true
6957 Babel.picture_has_bidi = 0
6958 }%
6959 \setbox\z@\hb@xt@z@{%
6960 \@defaultunitsset\@tempdimc{#1}\unitlength
6961 \kern\@tempdimc
6962 #3\hss}% TODO: #3 executed twice (below). That's bad.
6963 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6964 \fi
6965 % Do:
6966 \@defaultunitsset\@tempdimc{#2}\unitlength
6967 \raise\@tempdimc\hb@xt@z@{%
6968 \@defaultunitsset\@tempdimc{#1}\unitlength
6969 \kern\@tempdimc
6970 {\ifnum\bbl@tempc>z@\bbl@pictresetdir\fi#3}\hss}%
6971 \ignorespaces}%
6972 \MakeRobust\put}%
6973 \AtBeginDocument
6974 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir@gobble}%
6975 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?

```

```

6976 \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6977 \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6978 \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6979 \fi
6980 \ifx\tikzpicture\@undefined\else
6981 \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6982 \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6983 \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6984 \bbl@sreplace\tikzpicture{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6985 \fi
6986 \ifx\tcolorbox\@undefined\else
6987 \def\tcb@drawing@env@begin{%
6988 \csname tcb@before@\tcb@split@state\endcsname
6989 \bbl@pictsetdir\tw@
6990 \begin{\kvtcb@graphenv}%
6991 \tcb@bbdraw
6992 \tcb@apply@graph@patches}%
6993 \def\tcb@drawing@env@end{%
6994 \end{\kvtcb@graphenv}%
6995 \bbl@pictresetdir
6996 \csname tcb@after@\tcb@split@state\endcsname}%
6997 \fi
6998 }}
6999 {}

```

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`.

```

7000 \IfBabelLayout{counters*}%
7001 {\bbl@add\bbl@opt@layout{.counters.}%
7002 \directlua{
7003 \lua{
7004 \lua{
7005 }{}
7006 \IfBabelLayout{counters}%
7007 {\let\bbl@0L@@textsuperscript\@textsuperscript
7008 \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7009 \let\bbl@0L@latin@arabic=\@arabic
7010 \let\bbl@0L@arabic\@arabic
7011 \def\@arabic#1{\babelsublr{\bbl@latin@arabic#1}}%
7012 \ifpackagewith{babel}{bidi=default}%
7013 {\let\bbl@asciroman=\@roman
7014 \let\bbl@0L@@roman\@roman
7015 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7016 \let\bbl@asciRoman=\@Roman
7017 \let\bbl@0L@@roman\@Roman
7018 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
7019 \let\bbl@0L@labelenumii\labelenumii
7020 \def\labelenumii{\theenumii}%
7021 \let\bbl@0L@p@enumiii\p@enumiii
7022 \def\p@enumiii{\p@enumii}\theenumii{}}{}{}
7023 <@Footnote changes@>
7024 \IfBabelLayout{footnotes}%
7025 {\let\bbl@0L@footnote\footnote
7026 \BabelFootnote\footnote\language{}{}}%
7027 \BabelFootnote\localfootnote\language{}{}}%
7028 \BabelFootnote\mainfootnote{}{}}{}
7029 {}

```

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.

```

7030 \IfBabelLayout{extras}%
7031 {\bbl@ncarg\let\bbl@0L@underline{underline }%
7032 \bbl@carg\bbl@sreplace{underline }%

```



```

7033     {\@@@underline}{\bgroup\bbl@nextfake$\@@@underline}%
7034     \bbl@carg\bbl@sreplace{underline }%
7035     {\m@th$}{\m@th$\egroup}%
7036     \let\bbl@0L@LaTeXe\LaTeXe
7037     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7038       \if b\expandafter\@car\@series\@nil\boldmath\fi
7039       \babelsublr}%
7040       \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
7041   {}
7042 </luatex>

```

10.13 Lua: transforms

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

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

```

7043 <{*transforms}>
7044 Babel.linebreaking.replacements = {}
7045 Babel.linebreaking.replacements[0] = {} -- pre
7046 Babel.linebreaking.replacements[1] = {} -- post
7047
7048 function Babel.tovalue(v)
7049   if type(v) == 'table' then
7050     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7051   else
7052     return v
7053   end
7054 end
7055
7056 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7057
7058 function Babel.set_hboxed(head, gc)
7059   for item in node.traverse(head) do
7060     node.set_attribute(item, Babel.attr_hboxed, 1)
7061   end
7062   return head
7063 end
7064
7065 Babel.fetch_subtext = {}
7066
7067 Babel.ignore_pre_char = function(node)
7068   return (node.lang == Babel.nohyphenation)
7069 end
7070
7071 -- Merging both functions doesn't seem feasible, because there are too
7072 -- many differences.
7073 Babel.fetch_subtext[0] = function(head)
7074   local word_string = ''
7075   local word_nodes = {}
7076   local lang
7077   local item = head
7078   local inmath = false
7079
7080   while item do
7081

```

```

7082     if item.id == 11 then
7083         inmath = (item.subtype == 0)
7084     end
7085
7086     if inmath then
7087         -- pass
7088
7089     elseif item.id == 29 then
7090         local locale = node.get_attribute(item, Babel.attr_locale)
7091
7092         if lang == locale or lang == nil then
7093             lang = lang or locale
7094             if Babel.ignore_pre_char(item) then
7095                 word_string = word_string .. Babel.us_char
7096             else
7097                 if node.has_attribute(item, Babel.attr_hboxed) then
7098                     word_string = word_string .. Babel.us_char
7099                 else
7100                     word_string = word_string .. unicode.utf8.char(item.char)
7101                 end
7102             end
7103             word_nodes[#word_nodes+1] = item
7104         else
7105             break
7106         end
7107
7108     elseif item.id == 12 and item.subtype == 13 then
7109         if node.has_attribute(item, Babel.attr_hboxed) then
7110             word_string = word_string .. Babel.us_char
7111         else
7112             word_string = word_string .. ' '
7113         end
7114         word_nodes[#word_nodes+1] = item
7115
7116         -- Ignore leading unrecognized nodes, too.
7117     elseif word_string ~= '' then
7118         word_string = word_string .. Babel.us_char
7119         word_nodes[#word_nodes+1] = item -- Will be ignored
7120     end
7121
7122     item = item.next
7123 end
7124
7125 -- Here and above we remove some trailing chars but not the
7126 -- corresponding nodes. But they aren't accessed.
7127 if word_string:sub(-1) == ' ' then
7128     word_string = word_string:sub(1,-2)
7129 end
7130 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7131 return word_string, word_nodes, item, lang
7132 end
7133
7134 Babel.fetch_subtext[1] = function(head)
7135     local word_string = ''
7136     local word_nodes = {}
7137     local lang
7138     local item = head
7139     local inmath = false
7140
7141     while item do
7142
7143         if item.id == 11 then
7144             inmath = (item.subtype == 0)

```

```

7145     end
7146
7147     if inmath then
7148         -- pass
7149
7150     elseif item.id == 29 then
7151         if item.lang == lang or lang == nil then
7152             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7153                 lang = lang or item.lang
7154                 if node.has_attribute(item, Babel.attr_hboxed) then
7155                     word_string = word_string .. Babel.us_char
7156                 else
7157                     word_string = word_string .. unicode.utf8.char(item.char)
7158                 end
7159                 word_nodes[#word_nodes+1] = item
7160             end
7161         else
7162             break
7163         end
7164
7165     elseif item.id == 7 and item.subtype == 2 then
7166         if node.has_attribute(item, Babel.attr_hboxed) then
7167             word_string = word_string .. Babel.us_char
7168         else
7169             word_string = word_string .. '='
7170         end
7171         word_nodes[#word_nodes+1] = item
7172
7173     elseif item.id == 7 and item.subtype == 3 then
7174         if node.has_attribute(item, Babel.attr_hboxed) then
7175             word_string = word_string .. Babel.us_char
7176         else
7177             word_string = word_string .. '|'
7178         end
7179         word_nodes[#word_nodes+1] = item
7180
7181         -- (1) Go to next word if nothing was found, and (2) implicitly
7182         -- remove leading USs.
7183         elseif word_string == '' then
7184             -- pass
7185
7186         -- This is the responsible for splitting by words.
7187         elseif (item.id == 12 and item.subtype == 13) then
7188             break
7189
7190         else
7191             word_string = word_string .. Babel.us_char
7192             word_nodes[#word_nodes+1] = item -- Will be ignored
7193         end
7194
7195         item = item.next
7196     end
7197
7198     word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7199     return word_string, word_nodes, item, lang
7200 end
7201
7202 function Babel.pre_hyphenate_replace(head)
7203     Babel.hyphenate_replace(head, 0)
7204 end
7205
7206 function Babel.post_hyphenate_replace(head)
7207     Babel.hyphenate_replace(head, 1)

```

```

7208 end
7209
7210 Babel.us_char = string.char(31)
7211
7212 function Babel.hyphenate_replace(head, mode)
7213     local u = unicode.utf8
7214     local lbkr = Babel.linebreaking.replacements[mode]
7215     local tovalue = Babel.tovalue
7216
7217     local word_head = head
7218
7219     while true do -- for each subtext block
7220
7221         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7222
7223         if Babel.debug then
7224             print()
7225             print((mode == 0) and '@@@<' or '@@@>', w)
7226         end
7227
7228         if nw == nil and w == '' then break end
7229
7230         if not lang then goto next end
7231         if not lbkr[lang] then goto next end
7232
7233         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7234         -- loops are nested.
7235         for k=1, #lbkr[lang] do
7236             local p = lbkr[lang][k].pattern
7237             local r = lbkr[lang][k].replace
7238             local attr = lbkr[lang][k].attr or -1
7239
7240             if Babel.debug then
7241                 print('*****', p, mode)
7242             end
7243
7244             -- This variable is set in some cases below to the first *byte*
7245             -- after the match, either as found by u.match (faster) or the
7246             -- computed position based on sc if w has changed.
7247             local last_match = 0
7248             local step = 0
7249
7250             -- For every match.
7251             while true do
7252                 if Babel.debug then
7253                     print('====')
7254                 end
7255                 local new -- used when inserting and removing nodes
7256                 local dummy_node -- used by after
7257
7258                 local matches = { u.match(w, p, last_match) }
7259
7260                 if #matches < 2 then break end
7261
7262                 -- Get and remove empty captures (with ()'s, which return a
7263                 -- number with the position), and keep actual captures
7264                 -- (from (...)), if any, in matches.
7265                 local first = table.remove(matches, 1)
7266                 local last = table.remove(matches, #matches)
7267                 -- Non re-fetched substrings may contain \31, which separates
7268                 -- subsubstrings.
7269                 if string.find(w:sub(first, last-1), Babel.us_char) then break end
7270

```

```

7271     local save_last = last -- with A()BC()D, points to D
7272
7273     -- Fix offsets, from bytes to unicode. Explained above.
7274     first = u.len(w:sub(1, first-1)) + 1
7275     last = u.len(w:sub(1, last-1)) -- now last points to C
7276
7277     -- This loop stores in a small table the nodes
7278     -- corresponding to the pattern. Used by 'data' to provide a
7279     -- predictable behavior with 'insert' (w_nodes is modified on
7280     -- the fly), and also access to 'remove'd nodes.
7281     local sc = first-1 -- Used below, too
7282     local data_nodes = {}
7283
7284     local enabled = true
7285     for q = 1, last-first+1 do
7286         data_nodes[q] = w_nodes[sc+q]
7287         if enabled
7288             and attr > -1
7289             and not node.has_attribute(data_nodes[q], attr)
7290         then
7291             enabled = false
7292         end
7293     end
7294
7295     -- This loop traverses the matched substring and takes the
7296     -- corresponding action stored in the replacement list.
7297     -- sc = the position in substr nodes / string
7298     -- rc = the replacement table index
7299     local rc = 0
7300
7301     ----- TODO. dummy_node?
7302     while rc < last-first+1 or dummy_node do -- for each replacement
7303         if Babel.debug then
7304             print('.....', rc + 1)
7305         end
7306         sc = sc + 1
7307         rc = rc + 1
7308
7309         if Babel.debug then
7310             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7311             local ss = ''
7312             for itt in node.traverse(head) do
7313                 if itt.id == 29 then
7314                     ss = ss .. unicode.utf8.char(itt.char)
7315                 else
7316                     ss = ss .. '{' .. itt.id .. '}'
7317                 end
7318             end
7319             print('*****', ss)
7320
7321         end
7322
7323         local crep = r[rc]
7324         local item = w_nodes[sc]
7325         local item_base = item
7326         local placeholder = Babel.us_char
7327         local d
7328
7329         if crep and crep.data then
7330             item_base = data_nodes[crep.data]
7331         end
7332
7333         if crep then

```

```

7334         step = crep.step or step
7335     end
7336
7337     if crep and crep.after then
7338         crep.insert = true
7339         if dummy_node then
7340             item = dummy_node
7341         else -- TODO. if there is a node after?
7342             d = node.copy(item_base)
7343             head, item = node.insert_after(head, item, d)
7344             dummy_node = item
7345         end
7346     end
7347
7348     if crep and not crep.after and dummy_node then
7349         node.remove(head, dummy_node)
7350         dummy_node = nil
7351     end
7352
7353     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7354         if step == 0 then
7355             last_match = save_last -- Optimization
7356         else
7357             last_match = utf8.offset(w, sc+step)
7358         end
7359         goto next
7360
7361     elseif crep == nil or crep.remove then
7362         node.remove(head, item)
7363         table.remove(w_nodes, sc)
7364         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7365         sc = sc - 1 -- Nothing has been inserted.
7366         last_match = utf8.offset(w, sc+1+step)
7367         goto next
7368
7369     elseif crep and crep.kashida then -- Experimental
7370         node.set_attribute(item,
7371             Babel.attr_kashida,
7372             crep.kashida)
7373         last_match = utf8.offset(w, sc+1+step)
7374         goto next
7375
7376     elseif crep and crep.string then
7377         local str = crep.string(matches)
7378         if str == '' then -- Gather with nil
7379             node.remove(head, item)
7380             table.remove(w_nodes, sc)
7381             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7382             sc = sc - 1 -- Nothing has been inserted.
7383         else
7384             local loop_first = true
7385             for s in string.utfvalues(str) do
7386                 d = node.copy(item_base)
7387                 d.char = s
7388                 if loop_first then
7389                     loop_first = false
7390                     head, new = node.insert_before(head, item, d)
7391                     if sc == 1 then
7392                         word_head = head
7393                     end
7394                     w_nodes[sc] = d
7395                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7396                 else

```

```

7397         sc = sc + 1
7398         head, new = node.insert_before(head, item, d)
7399         table.insert(w_nodes, sc, new)
7400         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7401     end
7402     if Babel.debug then
7403         print('.....', 'str')
7404         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7405     end
7406     end -- for
7407     node.remove(head, item)
7408 end -- if ''
7409 last_match = utf8.offset(w, sc+1+step)
7410 goto next
7411
7412 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7413     d = node.new(7, 3) -- (disc, regular)
7414     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7415     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7416     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7417     d.attr = item_base.attr
7418     if crep.pre == nil then -- TeXbook p96
7419         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7420     else
7421         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7422     end
7423     placeholder = '|'
7424     head, new = node.insert_before(head, item, d)
7425
7426 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7427     -- ERROR
7428
7429 elseif crep and crep.penalty then
7430     d = node.new(14, 0) -- (penalty, userpenalty)
7431     d.attr = item_base.attr
7432     d.penalty = tovalue(crep.penalty)
7433     head, new = node.insert_before(head, item, d)
7434
7435 elseif crep and crep.space then
7436     -- 655360 = 10 pt = 10 * 65536 sp
7437     d = node.new(12, 13) -- (glue, spaceskip)
7438     local quad = font.getfont(item_base.font).size or 655360
7439     node.setglue(d, tovalue(crep.space[1]) * quad,
7440                 tovalue(crep.space[2]) * quad,
7441                 tovalue(crep.space[3]) * quad)
7442     if mode == 0 then
7443         placeholder = ' '
7444     end
7445     head, new = node.insert_before(head, item, d)
7446
7447 elseif crep and crep.norule then
7448     -- 655360 = 10 pt = 10 * 65536 sp
7449     d = node.new(2, 3) -- (rule, empty) = \no*rule
7450     local quad = font.getfont(item_base.font).size or 655360
7451     d.width = tovalue(crep.norule[1]) * quad
7452     d.height = tovalue(crep.norule[2]) * quad
7453     d.depth = tovalue(crep.norule[3]) * quad
7454     head, new = node.insert_before(head, item, d)
7455
7456 elseif crep and crep.spacefactor then
7457     d = node.new(12, 13) -- (glue, spaceskip)
7458     local base_font = font.getfont(item_base.font)
7459     node.setglue(d,

```

```

7460         tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7461         tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7462         tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7463     if mode == 0 then
7464         placeholder = ' '
7465     end
7466     head, new = node.insert_before(head, item, d)
7467
7468 elseif mode == 0 and crep and crep.space then
7469     -- ERROR
7470
7471 elseif crep and crep.kern then
7472     d = node.new(13, 1) -- (kern, user)
7473     local quad = font.getfont(item_base.font).size or 655360
7474     d.attr = item_base.attr
7475     d.kern = tovalue(crep.kern) * quad
7476     head, new = node.insert_before(head, item, d)
7477
7478 elseif crep and crep.node then
7479     d = node.new(crep.node[1], crep.node[2])
7480     d.attr = item_base.attr
7481     head, new = node.insert_before(head, item, d)
7482
7483 end -- i.e., replacement cases
7484
7485 -- Shared by disc, space(factor), kern, node and penalty.
7486 if sc == 1 then
7487     word_head = head
7488 end
7489 if crep.insert then
7490     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7491     table.insert(w_nodes, sc, new)
7492     last = last + 1
7493 else
7494     w_nodes[sc] = d
7495     node.remove(head, item)
7496     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7497 end
7498
7499 last_match = utf8.offset(w, sc+1+step)
7500
7501 ::next::
7502
7503 end -- for each replacement
7504
7505 if Babel.debug then
7506     print('.....', '/')
7507     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7508 end
7509
7510 if dummy_node then
7511     node.remove(head, dummy_node)
7512     dummy_node = nil
7513 end
7514
7515 end -- for match
7516
7517 end -- for patterns
7518
7519 ::next::
7520 word_head = nw
7521 end -- for substring
7522 return head

```



```

7523 end
7524
7525 -- This table stores capture maps, numbered consecutively
7526 Babel.capture_maps = {}
7527
7528 -- The following functions belong to the next macro
7529 function Babel.capture_func(key, cap)
7530     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
7531     local cnt
7532     local u = unicode.utf8
7533     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7534     if cnt == 0 then
7535         ret = u.gsub(ret, '{{(%%X%%X%%X+}}',
7536             function (n)
7537                 return u.char(tonumber(n, 16))
7538             end)
7539     end
7540     ret = ret:gsub("%[%[%]]%.", '')
7541     ret = ret:gsub("%.%[%[%]]%", '')
7542     return key .. "[=function(m) return ]] .. ret .. [[ end]]
7543 end
7544
7545 function Babel.capt_map(from, mapno)
7546     return Babel.capture_maps[mapno][from] or from
7547 end
7548
7549 -- Handle the {n|abc|ABC} syntax in captures
7550 function Babel.capture_func_map(capno, from, to)
7551     local u = unicode.utf8
7552     from = u.gsub(from, '{{(%%X%%X%%X+}}',
7553         function (n)
7554             return u.char(tonumber(n, 16))
7555         end)
7556     to = u.gsub(to, '{{(%%X%%X%%X+}}',
7557         function (n)
7558             return u.char(tonumber(n, 16))
7559         end)
7560     local froms = {}
7561     for s in string.utfcharacters(from) do
7562         table.insert(froms, s)
7563     end
7564     local cnt = 1
7565     table.insert(Babel.capture_maps, {})
7566     local mlen = table.getn(Babel.capture_maps)
7567     for s in string.utfcharacters(to) do
7568         Babel.capture_maps[mlen][froms[cnt]] = s
7569         cnt = cnt + 1
7570     end
7571     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7572         (mlen) .. ").. " .. "["
7573 end
7574
7575 -- Create/Extend reversed sorted list of kashida weights:
7576 function Babel.capture_kashida(key, wt)
7577     wt = tonumber(wt)
7578     if Babel.kashida_wts then
7579         for p, q in ipairs(Babel.kashida_wts) do
7580             if wt == q then
7581                 break
7582             elseif wt > q then
7583                 table.insert(Babel.kashida_wts, p, wt)
7584                 break
7585             elseif table.getn(Babel.kashida_wts) == p then

```

```

7586         table.insert(Babel.kashida_wts, wt)
7587     end
7588 end
7589 else
7590     Babel.kashida_wts = { wt }
7591 end
7592 return 'kashida = ' .. wt
7593 end
7594
7595 function Babel.capture_node(id, subtype)
7596     local sbt = 0
7597     for k, v in pairs(node.subtypes(id)) do
7598         if v == subtype then sbt = k end
7599     end
7600     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7601 end
7602
7603 -- Experimental: applies prehyphenation transforms to a string (letters
7604 -- and spaces).
7605 function Babel.string_prehyphenation(str, locale)
7606     local n, head, last, res
7607     head = node.new(8, 0) -- dummy (hack just to start)
7608     last = head
7609     for s in string.utfvalues(str) do
7610         if s == 20 then
7611             n = node.new(12, 0)
7612         else
7613             n = node.new(29, 0)
7614             n.char = s
7615         end
7616         node.set_attribute(n, Babel.attr_locale, locale)
7617         last.next = n
7618         last = n
7619     end
7620     head = Babel.hyphenate_replace(head, 0)
7621     res = ''
7622     for n in node.traverse(head) do
7623         if n.id == 12 then
7624             res = res .. ' '
7625         elseif n.id == 29 then
7626             res = res .. unicode.utf8.char(n.char)
7627         end
7628     end
7629     tex.print(res)
7630 end
7631 </transforms>

```

10.14 Lua: Auto bidi with basic and basic-r

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

```

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

```

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

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

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

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

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

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

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

```

7632 (*basic-r)
7633 Babel.bidi_enabled = true
7634
7635 require('babel-data-bidi.lua')
7636
7637 local characters = Babel.characters
7638 local ranges = Babel.ranges
7639
7640 local DIR = node.id("dir")
7641
7642 local function dir_mark(head, from, to, outer)
7643   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7644   local d = node.new(DIR)
7645   d.dir = '+' .. dir
7646   node.insert_before(head, from, d)
7647   d = node.new(DIR)
7648   d.dir = '-' .. dir
7649   node.insert_after(head, to, d)
7650 end
7651
7652 function Babel.bidi(head, ispar)
7653   local first_n, last_n          -- first and last char with nums
7654   local last_es                  -- an auxiliary 'last' used with nums
7655   local first_d, last_d          -- first and last char in L/R block
7656   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):

```

7657   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7658   local strong_lr = (strong == 'l') and 'l' or 'r'
7659   local outer = strong
7660
7661   local new_dir = false
7662   local first_dir = false
7663   local inmath = false
7664
7665   local last_lr
7666
7667   local type_n = ''
7668

```

```

7669 for item in node.traverse(head) do
7670
7671   -- three cases: glyph, dir, otherwise
7672   if item.id == node.id'glyph'
7673     or (item.id == 7 and item.subtype == 2) then
7674
7675     local itemchar
7676     if item.id == 7 and item.subtype == 2 then
7677       itemchar = item.replace.char
7678     else
7679       itemchar = item.char
7680     end
7681     local chardata = characters[itemchar]
7682     dir = chardata and chardata.d or nil
7683     if not dir then
7684       for nn, et in ipairs(ranges) do
7685         if itemchar < et[1] then
7686           break
7687         elseif itemchar <= et[2] then
7688           dir = et[3]
7689           break
7690         end
7691       end
7692     end
7693     dir = dir or 'l'
7694     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).

```

7695   if new_dir then
7696     attr_dir = 0
7697     for at in node.traverse(item.attr) do
7698       if at.number == Babel.attr_dir then
7699         attr_dir = at.value & 0x3
7700       end
7701     end
7702     if attr_dir == 1 then
7703       strong = 'r'
7704     elseif attr_dir == 2 then
7705       strong = 'al'
7706     else
7707       strong = 'l'
7708     end
7709     strong_lr = (strong == 'l') and 'l' or 'r'
7710     outer = strong_lr
7711     new_dir = false
7712   end
7713
7714   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7715   dir_real = dir -- We need dir_real to set strong below
7716   if dir == 'al' then dir = 'r' end -- W3

```

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

```

7717   if strong == 'al' then
7718     if dir == 'en' then dir = 'an' end -- W2
7719     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7720     strong_lr = 'r' -- W3
7721   end

```

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

```

7722     elseif item.id == node.id'dir' and not inmath then
7723         new_dir = true
7724         dir = nil
7725     elseif item.id == node.id'math' then
7726         inmath = (item.subtype == 0)
7727     else
7728         dir = nil          -- Not a char
7729     end

```

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

```

7730     if dir == 'en' or dir == 'an' or dir == 'et' then
7731         if dir ~= 'et' then
7732             type_n = dir
7733         end
7734         first_n = first_n or item
7735         last_n = last_es or item
7736         last_es = nil
7737     elseif dir == 'es' and last_n then -- W3+W6
7738         last_es = item
7739     elseif dir == 'cs' then          -- it's right - do nothing
7740     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7741         if strong_lr == 'r' and type_n ~= '' then
7742             dir_mark(head, first_n, last_n, 'r')
7743         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7744             dir_mark(head, first_n, last_n, 'r')
7745             dir_mark(head, first_d, last_d, outer)
7746             first_d, last_d = nil, nil
7747         elseif strong_lr == 'l' and type_n ~= '' then
7748             last_d = last_n
7749         end
7750         type_n = ''
7751         first_n, last_n = nil, nil
7752     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:

```

7753     if dir == 'l' or dir == 'r' then
7754         if dir ~= outer then
7755             first_d = first_d or item
7756             last_d = item
7757         elseif first_d and dir ~= strong_lr then
7758             dir_mark(head, first_d, last_d, outer)
7759             first_d, last_d = nil, nil
7760         end
7761     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.

```

7762     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7763         item.char = characters[item.char] and
7764             characters[item.char].m or item.char
7765     elseif (dir or new_dir) and last_lr ~= item then
7766         local mir = outer .. strong_lr .. (dir or outer)

```

```

7767     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7768         for ch in node.traverse(node.next(last_lr)) do
7769             if ch == item then break end
7770             if ch.id == node.id'glyph' and characters[ch.char] then
7771                 ch.char = characters[ch.char].m or ch.char
7772             end
7773         end
7774     end
7775 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).

```

7776     if dir == 'l' or dir == 'r' then
7777         last_lr = item
7778         strong = dir_real          -- Don't search back - best save now
7779         strong_lr = (strong == 'l') and 'l' or 'r'
7780     elseif new_dir then
7781         last_lr = nil
7782     end
7783 end

```

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

```

7784     if last_lr and outer == 'r' then
7785         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7786             if characters[ch.char] then
7787                 ch.char = characters[ch.char].m or ch.char
7788             end
7789         end
7790     end
7791     if first_n then
7792         dir_mark(head, first_n, last_n, outer)
7793     end
7794     if first_d then
7795         dir_mark(head, first_d, last_d, outer)
7796     end

```

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

```

7797     return node.prev(head) or head
7798 end
7799 </basic-r>

```

And here the Lua code for bidi=basic:

```

7800 <(*basic)
7801 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7802
7803 Babel.fontmap = Babel.fontmap or {}
7804 Babel.fontmap[0] = {}      -- l
7805 Babel.fontmap[1] = {}      -- r
7806 Babel.fontmap[2] = {}      -- al/an
7807
7808 -- To cancel mirroring. Also OML, OMS, U?
7809 Babel.symbol_fonts = Babel.symbol_fonts or {}
7810 Babel.symbol_fonts[font.id('tenln')] = true
7811 Babel.symbol_fonts[font.id('tenlnw')] = true
7812 Babel.symbol_fonts[font.id('tencirc')] = true
7813 Babel.symbol_fonts[font.id('tencircw')] = true
7814
7815 Babel.bidi_enabled = true
7816 Babel.mirroring_enabled = true
7817
7818 require('babel-data-bidi.lua')
7819
7820 local characters = Babel.characters

```

```

7821 local ranges = Babel.ranges
7822
7823 local DIR = node.id('dir')
7824 local GLYPH = node.id('glyph')
7825
7826 local function insert_implicit(head, state, outer)
7827   local new_state = state
7828   if state.sim and state.eim and state.sim ~= state.eim then
7829     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7830     local d = node.new(DIR)
7831     d.dir = '+' .. dir
7832     node.insert_before(head, state.sim, d)
7833     local d = node.new(DIR)
7834     d.dir = '-' .. dir
7835     node.insert_after(head, state.eim, d)
7836   end
7837   new_state.sim, new_state.eim = nil, nil
7838   return head, new_state
7839 end
7840
7841 local function insert_numeric(head, state)
7842   local new
7843   local new_state = state
7844   if state.san and state.ean and state.san ~= state.ean then
7845     local d = node.new(DIR)
7846     d.dir = '+TLT'
7847     _, new = node.insert_before(head, state.san, d)
7848     if state.san == state.sim then state.sim = new end
7849     local d = node.new(DIR)
7850     d.dir = '-TLT'
7851     _, new = node.insert_after(head, state.ean, d)
7852     if state.ean == state.eim then state.eim = new end
7853   end
7854   new_state.san, new_state.ean = nil, nil
7855   return head, new_state
7856 end
7857
7858 local function glyph_not_symbol_font(node)
7859   if node.id == GLYPH then
7860     return not Babel.symbol_fonts[node.font]
7861   else
7862     return false
7863   end
7864 end
7865
7866 -- TODO - \hbox with an explicit dir can lead to wrong results
7867 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7868 -- was made to improve the situation, but the problem is the 3-dir
7869 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7870 -- well.
7871
7872 function Babel.bidi(head, ispar, hdir)
7873   local d -- d is used mainly for computations in a loop
7874   local prev_d = ''
7875   local new_d = false
7876
7877   local nodes = {}
7878   local outer_first = nil
7879   local inmath = false
7880
7881   local glue_d = nil
7882   local glue_i = nil
7883

```

```

7884 local has_en = false
7885 local first_et = nil
7886
7887 local has_hyperlink = false
7888
7889 local ATDIR = Babel.attr_dir
7890 local attr_d, temp
7891 local locale_d
7892
7893 local save_outer
7894 local locale_d = node.get_attribute(head, ATDIR)
7895 if locale_d then
7896     locale_d = locale_d & 0x3
7897     save_outer = (locale_d == 0 and 'l') or
7898                 (locale_d == 1 and 'r') or
7899                 (locale_d == 2 and 'al')
7900 elseif ispar then -- Or error? Shouldn't happen
7901     -- when the callback is called, we are just _after_ the box,
7902     -- and the textdir is that of the surrounding text
7903     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7904 else -- Empty box
7905     save_outer = ('TRT' == hdir) and 'r' or 'l'
7906 end
7907 local outer = save_outer
7908 local last = outer
7909 -- 'al' is only taken into account in the first, current loop
7910 if save_outer == 'al' then save_outer = 'r' end
7911
7912 local fontmap = Babel.fontmap
7913
7914 for item in node.traverse(head) do
7915
7916     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
7917     locale_d = node.get_attribute(item, ATDIR)
7918     node.set_attribute(item, ATDIR, 0x80)
7919
7920     -- In what follows, #node is the last (previous) node, because the
7921     -- current one is not added until we start processing the neutrals.
7922     -- three cases: glyph, dir, otherwise
7923     if glyph_not_symbol_font(item)
7924         or (item.id == 7 and item.subtype == 2) then
7925
7926         if locale_d == 0x80 then goto nextnode end
7927
7928         local d_font = nil
7929         local item_r
7930         if item.id == 7 and item.subtype == 2 then
7931             item_r = item.replace -- automatic discs have just 1 glyph
7932         else
7933             item_r = item
7934         end
7935
7936         local chardata = characters[item_r.char]
7937         d = chardata and chardata.d or nil
7938         if not d or d == 'nsm' then
7939             for nn, et in ipairs(ranges) do
7940                 if item_r.char < et[1] then
7941                     break
7942                 elseif item_r.char <= et[2] then
7943                     if not d then d = et[3]
7944                     elseif d == 'nsm' then d_font = et[3]
7945                     end
7946                     break

```



```

7947         end
7948     end
7949 end
7950 d = d or 'l'
7951
7952 -- A short 'pause' in bidi for mapfont
7953 -- %%% TODO. move if fontmap here
7954 d_font = d_font or d
7955 d_font = (d_font == 'l' and 0) or
7956           (d_font == 'nsm' and 0) or
7957           (d_font == 'r' and 1) or
7958           (d_font == 'al' and 2) or
7959           (d_font == 'an' and 2) or nil
7960 if d_font and fontmap and fontmap[d_font][item_r.font] then
7961     item_r.font = fontmap[d_font][item_r.font]
7962 end
7963
7964 if new_d then
7965     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7966     if inmath then
7967         attr_d = 0
7968     else
7969         attr_d = locale_d & 0x3
7970     end
7971     if attr_d == 1 then
7972         outer_first = 'r'
7973         last = 'r'
7974     elseif attr_d == 2 then
7975         outer_first = 'r'
7976         last = 'al'
7977     else
7978         outer_first = 'l'
7979         last = 'l'
7980     end
7981     outer = last
7982     has_en = false
7983     first_et = nil
7984     new_d = false
7985 end
7986
7987 if glue_d then
7988     if (d == 'l' and 'l' or 'r') ~= glue_d then
7989         table.insert(nodes, {glue_i, 'on', nil})
7990     end
7991     glue_d = nil
7992     glue_i = nil
7993 end
7994
7995 elseif item.id == DIR then
7996     d = nil
7997     new_d = true
7998
7999 elseif item.id == node.id'glue' and item.subtype == 13 then
8000     glue_d = d
8001     glue_i = item
8002     d = nil
8003
8004 elseif item.id == node.id'math' then
8005     inmath = (item.subtype == 0)
8006
8007 elseif item.id == 8 and item.subtype == 19 then
8008     has_hyperlink = true
8009

```

```

8010 else
8011     d = nil
8012 end
8013
8014 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8015 if last == 'al' and d == 'en' then
8016     d = 'an'           -- W3
8017 elseif last == 'al' and (d == 'et' or d == 'es') then
8018     d = 'on'           -- W6
8019 end
8020
8021 -- EN + CS/ES + EN      -- W4
8022 if d == 'en' and #nodes >= 2 then
8023     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8024         and nodes[#nodes-1][2] == 'en' then
8025         nodes[#nodes][2] = 'en'
8026     end
8027 end
8028
8029 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
8030 if d == 'an' and #nodes >= 2 then
8031     if (nodes[#nodes][2] == 'cs')
8032         and nodes[#nodes-1][2] == 'an' then
8033         nodes[#nodes][2] = 'an'
8034     end
8035 end
8036
8037 -- ET/EN                -- W5 + W7->l / W6->on
8038 if d == 'et' then
8039     first_et = first_et or (#nodes + 1)
8040 elseif d == 'en' then
8041     has_en = true
8042     first_et = first_et or (#nodes + 1)
8043 elseif first_et then    -- d may be nil here !
8044     if has_en then
8045         if last == 'l' then
8046             temp = 'l'    -- W7
8047         else
8048             temp = 'en'   -- W5
8049         end
8050     else
8051         temp = 'on'       -- W6
8052     end
8053     for e = first_et, #nodes do
8054         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8055     end
8056     first_et = nil
8057     has_en = false
8058 end
8059
8060 -- Force mathdir in math if ON (currently works as expected only
8061 -- with 'l')
8062
8063 if inmath and d == 'on' then
8064     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8065 end
8066
8067 if d then
8068     if d == 'al' then
8069         d = 'r'
8070         last = 'al'
8071     elseif d == 'l' or d == 'r' then
8072         last = d

```

```

8073     end
8074     prev_d = d
8075     table.insert(nodes, {item, d, outer_first})
8076 end
8077
8078 outer_first = nil
8079
8080 ::nextnode::
8081
8082 end -- for each node
8083
8084 -- TODO -- repeated here in case EN/ET is the last node. Find a
8085 -- better way of doing things:
8086 if first_et then      -- dir may be nil here !
8087     if has_en then
8088         if last == 'l' then
8089             temp = 'l'    -- W7
8090         else
8091             temp = 'en'   -- W5
8092         end
8093     else
8094         temp = 'on'      -- W6
8095     end
8096     for e = first_et, #nodes do
8097         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8098     end
8099 end
8100
8101 -- dummy node, to close things
8102 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8103
8104 ----- NEUTRAL -----
8105
8106 outer = save_outer
8107 last = outer
8108
8109 local first_on = nil
8110
8111 for q = 1, #nodes do
8112     local item
8113
8114     local outer_first = nodes[q][3]
8115     outer = outer_first or outer
8116     last = outer_first or last
8117
8118     local d = nodes[q][2]
8119     if d == 'an' or d == 'en' then d = 'r' end
8120     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8121
8122     if d == 'on' then
8123         first_on = first_on or q
8124     elseif first_on then
8125         if last == d then
8126             temp = d
8127         else
8128             temp = outer
8129         end
8130         for r = first_on, q - 1 do
8131             nodes[r][2] = temp
8132             item = nodes[r][1]    -- MIRRORING
8133             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8134                 and temp == 'r' and characters[item.char] then
8135                 local font_mode = ''

```

```

8136         if item.font > 0 and font.fonts[item.font].properties then
8137             font_mode = font.fonts[item.font].properties.mode
8138         end
8139         if font_mode ~= 'harf' and font_mode ~= 'plug' then
8140             item.char = characters[item.char].m or item.char
8141         end
8142     end
8143 end
8144     first_on = nil
8145 end
8146
8147     if d == 'r' or d == 'l' then last = d end
8148 end
8149
8150 ----- IMPLICIT, REORDER -----
8151
8152 outer = save_outer
8153 last = outer
8154
8155 local state = {}
8156 state.has_r = false
8157
8158 for q = 1, #nodes do
8159
8160     local item = nodes[q][1]
8161
8162     outer = nodes[q][3] or outer
8163
8164     local d = nodes[q][2]
8165
8166     if d == 'nsm' then d = last end          -- W1
8167     if d == 'en' then d = 'an' end
8168     local isdir = (d == 'r' or d == 'l')
8169
8170     if outer == 'l' and d == 'an' then
8171         state.san = state.san or item
8172         state.ean = item
8173     elseif state.san then
8174         head, state = insert_numeric(head, state)
8175     end
8176
8177     if outer == 'l' then
8178         if d == 'an' or d == 'r' then      -- im -> implicit
8179             if d == 'r' then state.has_r = true end
8180             state.sim = state.sim or item
8181             state.eim = item
8182         elseif d == 'l' and state.sim and state.has_r then
8183             head, state = insert_implicit(head, state, outer)
8184         elseif d == 'l' then
8185             state.sim, state.eim, state.has_r = nil, nil, false
8186         end
8187     else
8188         if d == 'an' or d == 'l' then
8189             if nodes[q][3] then -- nil except after an explicit dir
8190                 state.sim = item -- so we move sim 'inside' the group
8191             else
8192                 state.sim = state.sim or item
8193             end
8194             state.eim = item
8195         elseif d == 'r' and state.sim then
8196             head, state = insert_implicit(head, state, outer)
8197         elseif d == 'r' then
8198             state.sim, state.eim = nil, nil

```

```

8199     end
8200 end
8201
8202 if isdir then
8203     last = d          -- Don't search back - best save now
8204 elseif d == 'on' and state.san then
8205     state.san = state.san or item
8206     state.ean = item
8207 end
8208
8209 end
8210
8211 head = node.prev(head) or head
8212 % \end{macrocode}
8213 %
8214 % Now direction nodes has been distributed with relation to characters
8215 % and spaces, we need to take into account \TeX-specific elements in
8216 % the node list, to move them at an appropriate place. Firstly, with
8217 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8218 % that the latter are still discardable.
8219 %
8220 % \begin{macrocode}
8221 --- FIXES ---
8222 if has_hyperlink then
8223     local flag, linking = 0, 0
8224     for item in node.traverse(head) do
8225         if item.id == DIR then
8226             if item.dir == '+TRT' or item.dir == '+TLT' then
8227                 flag = flag + 1
8228             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8229                 flag = flag - 1
8230             end
8231             elseif item.id == 8 and item.subtype == 19 then
8232                 linking = flag
8233             elseif item.id == 8 and item.subtype == 20 then
8234                 if linking > 0 then
8235                     if item.prev.id == DIR and
8236                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8237                         d = node.new(DIR)
8238                         d.dir = item.prev.dir
8239                         node.remove(head, item.prev)
8240                         node.insert_after(head, item, d)
8241                     end
8242                 end
8243                 linking = 0
8244             end
8245         end
8246     end
8247
8248     for item in node.traverse_id(10, head) do
8249         local p = item
8250         local flag = false
8251         while p.prev and p.prev.id == 14 do
8252             flag = true
8253             p = p.prev
8254         end
8255         if flag then
8256             node.insert_before(head, p, node.copy(item))
8257             node.remove(head, item)
8258         end
8259     end
8260
8261     return head

```

```

8262 end
8263 function Babel.unset_atdir(head)
8264   local ATDIR = Babel.attr_dir
8265   for item in node.traverse(head) do
8266     node.set_attribute(item, ATDIR, 0x80)
8267   end
8268   return head
8269 end
8270 ⟨/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.

```

8271 ⟨*nil⟩
8272 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8273 \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.

```

8274 \ifx\l@nil\undefined
8275   \newlanguage\l@nil
8276   \namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8277   \let\bbl@elt\relax
8278   \edef\bbl@languages{% Add it to the list of languages
8279     \bbl@languages\bbl@elt{nil}{the\l@nil}{}}
8280 \fi

```

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

```

8281 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

`\captionnil`
`\datenil`

```

8282 \let\captionnil\@empty
8283 \let\datenil\@empty

```

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

```

8284 \def\bbl@inidata@nil{%
8285   \bbl@elt{identification}{tag.ini}{und}%
8286   \bbl@elt{identification}{load.level}{0}%
8287   \bbl@elt{identification}{charset}{utf8}%
8288   \bbl@elt{identification}{version}{1.0}%

```

```

8289 \bbl@elt{identification}{date}{2022-05-16}%
8290 \bbl@elt{identification}{name.local}{nil}%
8291 \bbl@elt{identification}{name.english}{nil}%
8292 \bbl@elt{identification}{name.babel}{nil}%
8293 \bbl@elt{identification}{tag.bcp47}{und}%
8294 \bbl@elt{identification}{language.tag.bcp47}{und}%
8295 \bbl@elt{identification}{tag.opentype}{dflt}%
8296 \bbl@elt{identification}{script.name}{Latin}%
8297 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8298 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8299 \bbl@elt{identification}{level}{1}%
8300 \bbl@elt{identification}{encodings}{}%
8301 \bbl@elt{identification}{derivate}{no}}
8302 \@namedef{bbl@tbc@nil}{und}
8303 \@namedef{bbl@lbc@nil}{und}
8304 \@namedef{bbl@casing@nil}{und} % TODO
8305 \@namedef{bbl@lotf@nil}{dflt}
8306 \@namedef{bbl@elname@nil}{nil}
8307 \@namedef{bbl@lname@nil}{nil}
8308 \@namedef{bbl@esname@nil}{Latin}
8309 \@namedef{bbl@sname@nil}{Latin}
8310 \@namedef{bbl@sbc@nil}{Latn}
8311 \@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.

```

8312 \ldf@finish{nil}
8313 </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.

```

8314 <<Compute Julian day>> ≡
8315 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8316 \def\bbl@cs@gregleap#1{%
8317   (\bbl@fpmo{#1}{4} == 0) &&
8318   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
8319 \def\bbl@cs@jd#1#2#3{% year, month, day
8320   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8321     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8322     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8323     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8324 <</Compute Julian day>>

```

13.1. Islamic

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

```

8325 <ca-islamic>
8326 \ExplSyntaxOn
8327 <@Compute Julian day@>
8328 % == islamic (default)
8329 Not yet implemented
8330 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8331 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8332   ((#3 + ceil(29.5 * (#2 - 1)) +
8333     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8334     1948439.5) - 1) }

```

```

8335 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8336 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8337 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8338 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8339 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8340 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8341   \edef\bbl@tempa{%
8342     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8343   \edef#5{%
8344     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8345   \edef#6{\fp_eval:n{
8346     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8347   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}

```

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

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

```

8348 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8349 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8350 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8351 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8352 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8353 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8354 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8355 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8356 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8357 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8358 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8359 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8360 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8361 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8362 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8363 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8364 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8365 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8366 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8367 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8368 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8369 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8370 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8371 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8372 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8373 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8374 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8375 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8376 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8377 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8378 65401,65431,65460,65490,65520}
8379 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8380 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8381 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8382 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8383   \ifnum#2>2014 \ifnum#2<2038
8384     \bbl@afterfi\expandafter\@gobble
8385   \fi\fi
8386   {\bbl@error{year-out-range}{2014-2038}{}}}%
8387   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8388     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8389   \count@\@ne
8390   \bbl@foreach\bbl@cs@umalqura@data{%
8391     \advance\count@\@ne
8392     \ifnum##1>\bbl@tempd\else

```



```

8393 \edef\bbl@tempe{\the\count@}%
8394 \edef\bbl@tempb{##1}%
8395 \fi}%
8396 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8397 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8398 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8399 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8400 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8401 \ExplSyntaxOff
8402 \bbl@add\bbl@precalendar{%
8403 \bbl@replace\bbl@ld@calendar{-civil}}}%
8404 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8405 \bbl@replace\bbl@ld@calendar{+}}}%
8406 \bbl@replace\bbl@ld@calendar{-}}}%
8407 </ca-islamic>

```

13.2. Hebrew

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

```

8408 <ca-hebrew>
8409 \newcount\bbl@cntcommon
8410 \def\bbl@remainder#1#2#3{%
8411 #3=#1\relax
8412 \divide #3 by #2\relax
8413 \multiply #3 by -#2\relax
8414 \advance #3 by #1\relax}%
8415 \newif\ifbbl@divisible
8416 \def\bbl@checkifdivisible#1#2{%
8417 {\countdef\tmp=0
8418 \bbl@remainder{#1}{#2}{\tmp}%
8419 \ifnum \tmp=0
8420 \global\bbl@divisibletrue
8421 \else
8422 \global\bbl@divisiblefalse
8423 \fi}}
8424 \newif\ifbbl@gregleap
8425 \def\bbl@ifgregleap#1{%
8426 \bbl@checkifdivisible{#1}{4}%
8427 \ifbbl@divisible
8428 \bbl@checkifdivisible{#1}{100}%
8429 \ifbbl@divisible
8430 \bbl@checkifdivisible{#1}{400}%
8431 \ifbbl@divisible
8432 \bbl@gregleaptrue
8433 \else
8434 \bbl@gregleapfalse
8435 \fi
8436 \else
8437 \bbl@gregleaptrue
8438 \fi
8439 \else
8440 \bbl@gregleapfalse
8441 \fi
8442 \ifbbl@gregleap}
8443 \def\bbl@gregdayspriormonths#1#2#3{%
8444 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8445 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8446 \bbl@ifgregleap{#2}%
8447 \ifnum #1 > 2
8448 \advance #3 by 1
8449 \fi

```

```

8450     \fi
8451     \global\bbl@cntcommon=#3}%
8452     #3=\bbl@cntcommon}
8453 \def\bbl@gregdaysprioryears#1#2{%
8454 {\countdef\tmpc=4
8455  \countdef\tmpb=2
8456  \tmpb=#1\relax
8457  \advance \tmpb by -1
8458  \tmpc=\tmpb
8459  \multiply \tmpc by 365
8460  #2=\tmpc
8461  \tmpc=\tmpb
8462  \divide \tmpc by 4
8463  \advance #2 by \tmpc
8464  \tmpc=\tmpb
8465  \divide \tmpc by 100
8466  \advance #2 by -\tmpc
8467  \tmpc=\tmpb
8468  \divide \tmpc by 400
8469  \advance #2 by \tmpc
8470  \global\bbl@cntcommon=#2\relax}%
8471  #2=\bbl@cntcommon}
8472 \def\bbl@absfromgreg#1#2#3#4{%
8473 {\countdef\tmpd=0
8474  #4=#1\relax
8475  \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8476  \advance #4 by \tmpd
8477  \bbl@gregdaysprioryears{#3}{\tmpd}%
8478  \advance #4 by \tmpd
8479  \global\bbl@cntcommon=#4\relax}%
8480  #4=\bbl@cntcommon}
8481 \newif\ifbbl@hebrleap
8482 \def\bbl@checkleaphebryear#1{%
8483 {\countdef\tmpa=0
8484  \countdef\tmpb=1
8485  \tmpa=#1\relax
8486  \multiply \tmpa by 7
8487  \advance \tmpa by 1
8488  \bbl@remainder{\tmpa}{19}{\tmpb}%
8489  \ifnum \tmpb < 7
8490    \global\bbl@hebrleaptrue
8491  \else
8492    \global\bbl@hebrleapfalse
8493  \fi}}
8494 \def\bbl@hebrrelapsedmonths#1#2{%
8495 {\countdef\tmpa=0
8496  \countdef\tmpb=1
8497  \countdef\tmpc=2
8498  \tmpa=#1\relax
8499  \advance \tmpa by -1
8500  #2=\tmpa
8501  \divide #2 by 19
8502  \multiply #2 by 235
8503  \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8504  \tmpc=\tmpb
8505  \multiply \tmpb by 12
8506  \advance #2 by \tmpb
8507  \multiply \tmpc by 7
8508  \advance \tmpc by 1
8509  \divide \tmpc by 19
8510  \advance #2 by \tmpc
8511  \global\bbl@cntcommon=#2}%
8512  #2=\bbl@cntcommon}

```

```

8513 \def\bbl@hebreleaseddays#1#2{%
8514   {\countdef\tmpa=0
8515    \countdef\tmpb=1
8516    \countdef\tmpc=2
8517    \bbl@hebreleasedmonths{#1}{#2}%
8518    \tmpa=#2\relax
8519    \multiply\tmpa by 13753
8520    \advance\tmpa by 5604
8521    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8522    \divide\tmpa by 25920
8523    \multiply#2 by 29
8524    \advance#2 by 1
8525    \advance#2 by \tmpa
8526    \bbl@remainder{#2}{7}{\tmpa}%
8527    \ifnum\tmpc < 19440
8528      \ifnum\tmpc < 9924
8529        \else
8530          \ifnum\tmpa=2
8531            \bbl@checkleaphebrewyear{#1}% of a common year
8532            \ifbbl@hebrleap
8533              \else
8534                \advance#2 by 1
8535              \fi
8536            \fi
8537          \fi
8538          \ifnum\tmpc < 16789
8539            \else
8540              \ifnum\tmpa=1
8541                \advance#1 by -1
8542                \bbl@checkleaphebrewyear{#1}% at the end of leap year
8543                \ifbbl@hebrleap
8544                  \advance#2 by 1
8545                \fi
8546              \fi
8547            \fi
8548          \else
8549            \advance#2 by 1
8550          \fi
8551          \bbl@remainder{#2}{7}{\tmpa}%
8552          \ifnum\tmpa=0
8553            \advance#2 by 1
8554          \else
8555            \ifnum\tmpa=3
8556              \advance#2 by 1
8557            \else
8558              \ifnum\tmpa=5
8559                \advance#2 by 1
8560              \fi
8561            \fi
8562          \fi
8563          \global\bbl@cntcommon=#2\relax}%
8564   #2=\bbl@cntcommon}
8565 \def\bbl@daysinhebrewyear#1#2{%
8566   {\countdef\tmpe=12
8567    \bbl@hebreleaseddays{#1}{\tmpe}%
8568    \advance#1 by 1
8569    \bbl@hebreleaseddays{#1}{#2}%
8570    \advance#2 by -\tmpe
8571    \global\bbl@cntcommon=#2}%
8572   #2=\bbl@cntcommon}
8573 \def\bbl@hebrdayspriormonths#1#2#3{%
8574   {\countdef\tmpf= 14
8575    #3=\ifcase #1

```

```

8576         0 \or
8577         0 \or
8578         30 \or
8579         59 \or
8580         89 \or
8581        118 \or
8582        148 \or
8583        148 \or
8584        177 \or
8585        207 \or
8586        236 \or
8587        266 \or
8588        295 \or
8589        325 \or
8590        400
8591    \fi
8592    \bbl@checkleaphebryear{#2}%
8593    \ifbbl@hebrleap
8594        \ifnum #1 > 6
8595            \advance #3 by 30
8596        \fi
8597    \fi
8598    \bbl@daysinhebryear{#2}{\tmpf}%
8599    \ifnum #1 > 3
8600        \ifnum \tmpf=353
8601            \advance #3 by -1
8602        \fi
8603        \ifnum \tmpf=383
8604            \advance #3 by -1
8605        \fi
8606    \fi
8607    \ifnum #1 > 2
8608        \ifnum \tmpf=355
8609            \advance #3 by 1
8610        \fi
8611        \ifnum \tmpf=385
8612            \advance #3 by 1
8613        \fi
8614    \fi
8615    \global\bbl@cntcommon=#3\relax}%
8616    #3=\bbl@cntcommon}
8617 \def\bbl@absfromhebr#1#2#3#4{%
8618 {#4=#1\relax
8619 \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8620 \advance #4 by #1\relax
8621 \bbl@hebrrelapseddays{#3}{#1}%
8622 \advance #4 by #1\relax
8623 \advance #4 by -1373429
8624 \global\bbl@cntcommon=#4\relax}%
8625 #4=\bbl@cntcommon}
8626 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8627 {\countdef\tmpx= 17
8628 \countdef\tmpy= 18
8629 \countdef\tmpz= 19
8630 #6=#3\relax
8631 \global\advance #6 by 3761
8632 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8633 \tmpz=1 \tmpy=1
8634 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8635 \ifnum \tmpx > #4\relax
8636     \global\advance #6 by -1
8637     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8638 \fi

```

```

8639 \advance #4 by -\tmpx
8640 \advance #4 by 1
8641 #5=#4\relax
8642 \divide #5 by 30
8643 \loop
8644 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8645 \ifnum \tmpx < #4\relax
8646 \advance #5 by 1
8647 \tmpy=\tmpx
8648 \repeat
8649 \global\advance #5 by -1
8650 \global\advance #4 by -\tmpy}}
8651 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8652 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8653 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8654 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8655 \bbl@hebrfromgreg
8656 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8657 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8658 \edef#4{\the\bbl@hebyear}%
8659 \edef#5{\the\bbl@hebrmonth}%
8660 \edef#6{\the\bbl@hebrday}}
8661 </ca-hebrew>

```

13.3. Persian

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

```

8662 <*ca-persian>
8663 \ExplSyntaxOn
8664 <@Compute Julian day@>
8665 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8666 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8667 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8668 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8669 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8670 \bbl@afterfi\expandafter\@gobble
8671 \fi\fi
8672 {\bbl@error{year-out-range}{2013-2050}{}}}%
8673 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8674 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8675 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8676 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8677 \ifnum\bbl@tempc<\bbl@tempb
8678 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8679 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8680 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8681 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8682 \fi
8683 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8684 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8685 \edef#5{\fp_eval:n{% set Jalali month
8686 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8687 \edef#6{\fp_eval:n{% set Jalali day
8688 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}
8689 \ExplSyntaxOff
8690 </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.

```

8691 < *ca-coptic >
8692 \ExplSyntaxOn
8693 <@Compute Julian day@>
8694 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8695   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8696   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8697   \edef#4{\fp_eval:n{%
8698     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8699   \edef\bbl@tempc{\fp_eval:n{%
8700     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8701   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8702   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8703 \ExplSyntaxOff
8704 < /ca-coptic >
8705 < *ca-ethiopic >
8706 \ExplSyntaxOn
8707 <@Compute Julian day@>
8708 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8709   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8710   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8711   \edef#4{\fp_eval:n{%
8712     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8713   \edef\bbl@tempc{\fp_eval:n{%
8714     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8715   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8716   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8717 \ExplSyntaxOff
8718 < /ca-ethiopic >

```

13.5. Buddhist

That's very simple.

```

8719 < *ca-buddhist >
8720 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8721   \edef#4{\number\numexpr#1+543\relax}%
8722   \edef#5{#2}%
8723   \edef#6{#3}}
8724 < /ca-buddhist >
8725 %
8726 % \subsection{Chinese}
8727 %
8728 % Brute force, with the Julian day of first day of each month. The
8729 % table has been computed with the help of \textsf{python-lunardate} by
8730 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8731 % is 2015-2044.
8732 %
8733 % \begin{macrocode}
8734 < *ca-chinese >
8735 \ExplSyntaxOn
8736 <@Compute Julian day@>
8737 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
8738   \edef\bbl@tempd{\fp_eval:n{%
8739     \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8740   \count@\z@
8741   \@tempcnta=2015
8742   \bbl@foreach\bbl@cs@chinese@data{%
8743     \ifnum##1>\bbl@tempd\else
8744       \advance\count@\@ne
8745     \ifnum\count@>12

```

```

8746 \count@\@ne
8747 \advance\@tempcnta\@ne\fi
8748 \bbl@xin@{,##1,},{, \bbl@cs@chinese@leap,}%
8749 \ifin@
8750 \advance\count@\m@ne
8751 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8752 \else
8753 \edef\bbl@tempe{\the\count@}%
8754 \fi
8755 \edef\bbl@tempb{##1}%
8756 \fi}%
8757 \edef#4{\the\@tempcnta}%
8758 \edef#5{\bbl@tempe}%
8759 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8760 \def\bbl@cs@chinese@leap{%
8761 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8762 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8763 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8764 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8765 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8766 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8767 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8768 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8769 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8770 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8771 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8772 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8773 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8774 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8775 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8776 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8777 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8778 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8779 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8780 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8781 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8782 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8783 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8784 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8785 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8786 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8787 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8788 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8789 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8790 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8791 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8792 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8793 10896,10926,10956,10986,11015,11045,11074,11103}
8794 \ExplSyntaxOff
8795 </ca-chinese>

```

14. Support for Plain T_EX (plain.def)

14.1. Not renaming hyphen.tex

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

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

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

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

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

```
8796 <*\bplain | blplain>
8797 \catcode`\{=1 % left brace is begin-group character
8798 \catcode`\}=2 % right brace is end-group character
8799 \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).

```
8800 \openin 0 hyphen.cfg
8801 \ifeof0
8802 \else
8803   \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.

```
8804   \def\input #1 {%
8805     \let\input\a
8806     \a hyphen.cfg
8807     \let\a\undefined
8808   }
8809 \fi
8810 </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`.

```
8811 <bplain>\a plain.tex
8812 <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.

```
8813 <bplain>\def\fmtname{babel-plain}
8814 <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).

```
8815 <<*\Emulate LaTeX>> ≡
8816 \def\@empty{}
8817 \def\loadlocalcfg#1{%
8818   \openin0#1.cfg
8819   \ifeof0
8820     \closein0
8821   \else
8822     \closein0
8823     {\immediate\writel6{*****}%
8824      \immediate\writel6{* Local config file #1.cfg used}%
8825      \immediate\writel6{**}%
8826     }
8827     \input #1.cfg\relax
8828   \fi
8829   \@endofldef}
```


14.3. General tools

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

```
8830 \long\def\@firstofone#1{#1}
8831 \long\def\@firstoftwo#1#2{#1}
8832 \long\def\@secondoftwo#1#2{#2}
8833 \def\@nnil{\@nil}
8834 \def\@gobbletwo#1#2{}
8835 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8836 \def\@star@or@long#1{%
8837   \@ifstar
8838   {\let\l@ngrel@x\relax#1}%
8839   {\let\l@ngrel@x\long#1}}
8840 \let\l@ngrel@x\relax
8841 \def\@car#1#2\@nil{#1}
8842 \def\@cdr#1#2\@nil{#2}
8843 \let\@typeset@protect\relax
8844 \let\protected@edef\edef
8845 \long\def\@gobble#1{}
8846 \edef\@backslashchar{\expandafter\@gobble\string\}
8847 \def\strip@prefix#1>{}
8848 \def\g@addto@macro#1#2{{%
8849   \toks@\expandafter{#1#2}%
8850   \xdef#1{\the\toks@}}}
8851 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8852 \def\@nameuse#1{\csname #1\endcsname}
8853 \def\@ifundefined#1{%
8854   \expandafter\ifx\csname#1\endcsname\relax
8855     \expandafter\@firstoftwo
8856     \else
8857       \expandafter\@secondoftwo
8858     \fi}
8859 \def\@expandtwoargs#1#2#3{%
8860   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8861 \def\zap@space#1 #2{%
8862   #1%
8863   \ifx#2\@empty\else\expandafter\zap@space\fi
8864   #2}
8865 \let\bbl@trace\@gobble
8866 \def\bbl@error#1{% Implicit #2#3#4
8867   \begingroup
8868     \catcode`\=0 \catcode`\==12 \catcode`\`=12
8869     \catcode`\^M=5 \catcode`\%=14
8870     \input errbabel.def
8871   \endgroup
8872   \bbl@error{#1}}
8873 \def\bbl@warning#1{%
8874   \begingroup
8875     \newlinechar=`^^J
8876     \def\{^J(babel) }%
8877     \message{\{^J#1}%
8878   \endgroup}
8879 \let\bbl@infowarn\bbl@warning
8880 \def\bbl@info#1{%
8881   \begingroup
8882     \newlinechar=`^^J
8883     \def\{^J}%
8884     \wlog{#1}%
8885   \endgroup}
```

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

```
8886 \ifx\@preamblecmds\undefined
```

```

8887 \def\@preamblecmds{}
8888 \fi
8889 \def\@onlypreamble#1{%
8890 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8891 \@preamblecmds\do#1}}
8892 \@onlypreamble\@onlypreamble

```

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

```

8893 \def\begindocument{%
8894 \@begindocumenthook
8895 \global\let\@begindocumenthook\@undefined
8896 \def\do##1{\global\let##1\@undefined}%
8897 \@preamblecmds
8898 \global\let\do\noexpand}
8899 \ifx\@begindocumenthook\@undefined
8900 \def\@begindocumenthook{}
8901 \fi
8902 \@onlypreamble\@begindocumenthook
8903 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8904 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8905 \@onlypreamble\AtEndOfPackage
8906 \def\@endofldf{}
8907 \@onlypreamble\@endofldf
8908 \let\bbl@afterlang\@empty
8909 \chardef\bbl@opt@hyphenmap\z@

```

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

```

8910 \catcode`\&=\z@
8911 \ifx\if@files\@undefined
8912 \expandafter\let\csname if@files\expandafter\endcsname
8913 \csname iffalse\endcsname
8914 \fi
8915 \catcode`\&=4

```

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

```

8916 \def\newcommand{\@star@or@long\new@command}
8917 \def\new@command#1{%
8918 \@testopt{\@newcommand#1}0}
8919 \def\@newcommand#1[#2]{%
8920 \@ifnextchar [{\@xargdef#1[#2]}%
8921 {\@argdef#1[#2]}}
8922 \long\def\@argdef#1[#2]#3{%
8923 \@yargdef#1\@ne{#2}{#3}}
8924 \long\def\@xargdef#1[#2][#3]#4{%
8925 \expandafter\def\expandafter#1\expandafter{%
8926 \expandafter\@protected@testopt\expandafter #1%
8927 \csname\string#1\expandafter\endcsname{#3}}}%
8928 \expandafter\@yargdef \csname\string#1\endcsname
8929 \tw@{#2}{#4}}
8930 \long\def\@yargdef#1#2#3{%
8931 \@tempcnta#3\relax
8932 \advance \@tempcnta \@ne
8933 \let\@hash@\relax
8934 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8935 \@tempcntb #2%
8936 \@whilenum \@tempcntb <\@tempcnta
8937 \do{%
8938 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%

```

```

8939 \advance\@tempcntb \@ne}%
8940 \let\@hash@##%
8941 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8942 \def\providecommand{\@star@or@long\provide@command}
8943 \def\provide@command#1{%
8944 \begingroup
8945 \escapechar\m@ne\xdef\@gtempa{\string#1}%
8946 \endgroup
8947 \expandafter\ifundefined\@gtempa
8948 {\def\reserved@a{\new@command#1}}%
8949 {\let\reserved@a\relax
8950 \def\reserved@a{\new@command\reserved@a}}%
8951 \reserved@a}%

8952 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8953 \def\declare@robustcommand#1{%
8954 \edef\reserved@a{\string#1}%
8955 \def\reserved@b{#1}%
8956 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8957 \edef#1{%
8958 \ifx\reserved@a\reserved@b
8959 \noexpand\x@protect
8960 \noexpand#1%
8961 \fi
8962 \noexpand\protect
8963 \expandafter\noexpand\csname
8964 \expandafter\@gobble\string#1 \endcsname
8965 }%
8966 \expandafter\new@command\csname
8967 \expandafter\@gobble\string#1 \endcsname
8968 }
8969 \def\x@protect#1{%
8970 \ifx\protect\@typeset@protect\else
8971 \@x@protect#1%
8972 \fi
8973 }
8974 \catcode`\&=\z@ % Trick to hide conditionals
8975 \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`.

```

8976 \def\bbl@tempa{\csname newif\endcsname&ifin@}
8977 \catcode`\&=4
8978 \ifx\in@\@undefined
8979 \def\in@#1#2{%
8980 \def\in@##1#1##2##3\in@{%
8981 \ifx\in@##2\in@false\else\in@true\fi}%
8982 \in@##2#1\in@\in@}
8983 \else
8984 \let\bbl@tempa\@empty
8985 \fi
8986 \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).

```

8987 \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.

```

8988 \def\@ifl@aded#1#2#3#4{}

```

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

```
8989 \ifx\@tempcnta\@undefined
8990   \csname newcount\endcsname\@tempcnta\relax
8991 \fi
8992 \ifx\@tempcntb\@undefined
8993   \csname newcount\endcsname\@tempcntb\relax
8994 \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`).

```
8995 \ifx\bye\@undefined
8996   \advance\count10 by -2\relax
8997 \fi
8998 \ifx\@ifnextchar\@undefined
8999   \def\@ifnextchar#1#2#3{%
9000     \let\reserved@d=#1%
9001     \def\reserved@a{#2}\def\reserved@b{#3}%
9002     \futurelet\@let@token\@ifnch}
9003 \def\@ifnch{%
9004   \ifx\@let@token\@sptoken
9005     \let\reserved@c\@xifnch
9006   \else
9007     \ifx\@let@token\reserved@d
9008       \let\reserved@c\reserved@a
9009     \else
9010       \let\reserved@c\reserved@b
9011     \fi
9012   \fi
9013   \reserved@c}
9014 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
9015 \def\:\@xifnch\ \expandafter\def\:{\futurelet\@let@token\@ifnch}
9016 \fi
9017 \def\@testopt#1#2{%
9018   \@ifnextchar[#{1}{#1[#{2]}}
9019 \def\@protected@testopt#1{%
9020   \ifx\protect\@typeset@protect
9021     \expandafter\@testopt
9022   \else
9023     \@x@protect#1%
9024   \fi}
9025 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9026   #2\relax}\fi}
9027 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9028   \else\expandafter\@gobble\fi{#1}}
```

14.4. Encoding related macros

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

```
9029 \def\DeclareTextCommand{%
9030   \@dec@text@cmd\providecommand
9031 }
9032 \def\ProvideTextCommand{%
9033   \@dec@text@cmd\providecommand
9034 }
9035 \def\DeclareTextSymbol#1#2#3{%
9036   \@dec@text@cmd\chardef#1{#2}#3\relax
9037 }
9038 \def\@dec@text@cmd#1#2#3{%
9039   \expandafter\def\expandafter#2%
9040     \expandafter{%
```

```

9041         \csname#3-cmd\expandafter\endcsname
9042         \expandafter#2%
9043         \csname#3\string#2\endcsname
9044     }%
9045 %   \let\@ifdefinable\@rc@ifdefinable
9046     \expandafter#1\csname#3\string#2\endcsname
9047 }
9048 \def\@current@cmd#1{%
9049     \ifx\protect\@typeset@protect\else
9050         \noexpand#1\expandafter\@gobble
9051     \fi
9052 }
9053 \def\@changed@cmd#1#2{%
9054     \ifx\protect\@typeset@protect
9055         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9056             \expandafter\ifx\csname ?\string#1\endcsname\relax
9057                 \expandafter\def\csname ?\string#1\endcsname{%
9058                     \@changed@x@err{#1}%
9059                 }%
9060             \fi
9061             \global\expandafter\let
9062                 \csname\cf@encoding \string#1\expandafter\endcsname
9063                 \csname ?\string#1\endcsname
9064             \fi
9065             \csname\cf@encoding\string#1%
9066             \expandafter\endcsname
9067         \else
9068             \noexpand#1%
9069         \fi
9070 }
9071 \def\@changed@x@err#1{%
9072     \errhelp{Your command will be ignored, type <return> to proceed}%
9073     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9074 \def\DeclareTextCommandDefault#1{%
9075     \DeclareTextCommand#1?%
9076 }
9077 \def\ProvideTextCommandDefault#1{%
9078     \ProvideTextCommand#1?%
9079 }
9080 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9081 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9082 \def\DeclareTextAccent#1#2#3{%
9083     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9084 }
9085 \def\DeclareTextCompositeCommand#1#2#3#4{%
9086     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9087     \edef\reserved@b{\string##1}%
9088     \edef\reserved@c{%
9089         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9090     \ifx\reserved@b\reserved@c
9091         \expandafter\expandafter\expandafter\ifx
9092             \expandafter\@car\reserved@a\relax\relax\@nil
9093             \@text@composite
9094         \else
9095             \edef\reserved@b##1{%
9096                 \def\expandafter\noexpand
9097                     \csname#2\string#1\endcsname###1{%
9098                     \noexpand\@text@composite
9099                     \expandafter\noexpand\csname#2\string#1\endcsname
9100                     ###1\noexpand\@empty\noexpand\@text@composite
9101                     {##1}%
9102                 }%
9103             }%

```

```

9104     \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9105     \fi
9106     \expandafter\def\csname\expandafter\string\csname
9107       #2\endcsname\string#1-\string#3\endcsname{#4}
9108   \else
9109     \errhelp{Your command will be ignored, type <return> to proceed}%
9110     \errmessage{\string\DeclareTextCompositeCommand\space used on
9111       inappropriate command \protect#1}
9112   \fi
9113 }
9114 \def\@text@composite#1#2#3\@text@composite{%
9115   \expandafter\@text@composite@x
9116     \csname\string#1-\string#2\endcsname
9117 }
9118 \def\@text@composite@x#1#2{%
9119   \ifx#1\relax
9120     #2%
9121   \else
9122     #1%
9123   \fi
9124 }
9125 %
9126 \def\@strip@args#1:#2-#3\@strip@args{#2}
9127 \def\DeclareTextComposite#1#2#3#4{%
9128   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9129   \bgroup
9130     \lccode`\@=#4%
9131     \lowercase{%
9132   \egroup
9133     \reserved@a \@%
9134   }%
9135 }
9136 %
9137 \def\UseTextSymbol#1#2{#2}
9138 \def\UseTextAccent#1#2#3{}
9139 \def\@use@text@encoding#1{}
9140 \def\DeclareTextSymbolDefault#1#2{%
9141   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9142 }
9143 \def\DeclareTextAccentDefault#1#2{%
9144   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9145 }
9146 \def\cf@encoding{OT1}

```

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

```

9147 \DeclareTextAccent{"}{OT1}{127}
9148 \DeclareTextAccent{'}{OT1}{19}
9149 \DeclareTextAccent{^}{OT1}{94}
9150 \DeclareTextAccent{\`}{OT1}{18}
9151 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9152 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9153 \DeclareTextSymbol{\textquotedblright}{OT1}{`\`}
9154 \DeclareTextSymbol{\textquoteleft}{OT1}{``}`
9155 \DeclareTextSymbol{\textquoteright}{OT1}{``}`
9156 \DeclareTextSymbol{\i}{OT1}{16}
9157 \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`.

```

9158 \ifx\scriptsize\undefined
9159   \let\scriptsize\sevenrm

```

```

9160 \fi

And a few more “dummy” definitions.

9161 \def\language{english}%
9162 \let\bbl@opt@shorthands\@nnil
9163 \def\bbl@ifshorthand#1#2#3{#2}%
9164 \let\bbl@language@opts\@empty
9165 \let\bbl@ensureinfo\@gobble
9166 \let\bbl@provide@locale\relax
9167 \ifx\babeloptionstrings\undefined
9168   \let\bbl@opt@strings\@nnil
9169 \else
9170   \let\bbl@opt@strings\babeloptionstrings
9171 \fi
9172 \def\BabelStringsDefault{generic}
9173 \def\bbl@tempa{normal}
9174 \ifx\babeloptionmath\bbl@tempa
9175   \def\bbl@mathnormal{\noexpand\textormath}
9176 \fi
9177 \def\AfterBabelLanguage#1#2{}
9178 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9179 \let\bbl@afterlang\relax
9180 \def\bbl@opt@safe{BR}
9181 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
9182 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9183 \expandafter\newif\csname ifbbl@single\endcsname
9184 \chardef\bbl@bidimode\z@
9185 <</Emulate LaTeX>>

A proxy file:

9186 <*\plain>
9187 \input babel.def
9188 </\plain>

```

15. Acknowledgements

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

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

Barbara Beeton has helped in improving the manual.

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

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \LaTeX styles*, *TUGboat* 10 (1989) #3, pp. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O’Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O’Reilly, 2006.
- [6] Leslie Lamport, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O’Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018

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