

Babel

Code

Version 25.13.102165
2025/10/17

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

LuaT_EX

pdfT_EX

XeT_EX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	A few core definitions	8
3.2	LaTeX: babel.sty (start)	8
3.3	base	9
3.4	key=value options and other general option	10
3.5	Post-process some options	11
3.6	Plain: babel.def (start)	13
4	babel.sty and babel.def (common)	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	More on selection	23
4.4	Short tags	25
4.5	Compatibility with language.def	25
4.6	Hooks	26
4.7	Setting up language files	27
4.8	Shorthands	29
4.9	Language attributes	38
4.10	Support for saving and redefining macros	39
4.11	French spacing	40
4.12	Hyphens	41
4.13	Multiencoding strings	43
4.14	Tailor captions	48
4.15	Making glyphs available	49
4.15.1	Quotation marks	49
4.15.2	Letters	50
4.15.3	Shorthands for quotation marks	51
4.15.4	Umlauts and tremas	52
4.16	Layout	53
4.17	Load engine specific macros	54
4.18	Creating and modifying languages	54
4.19	Main loop in ‘provide’	62
4.20	Processing keys in ini	66
4.21	French spacing (again)	72
4.22	Handle language system	73
4.23	Numerals	73
4.24	Casing	75
4.25	Getting info	76
4.26	BCP 47 related commands	77
5	Adjusting the Babel behavior	78
5.1	Cross referencing macros	80
5.2	Layout	83
5.3	Marks	84
5.4	Other packages	85
5.4.1	ifthen	85
5.4.2	varioref	86
5.4.3	hhline	86
5.5	Encoding and fonts	87
5.6	Basic bidi support	88
5.7	Local Language Configuration	92
5.8	Language options	92

6	The kernel of Babel	96
7	Error messages	96
8	Loading hyphenation patterns	100
9	luatex + xetex: common stuff	104
10	Hooks for XeTeX and LuaTeX	107
10.1	XeTeX	107
10.2	Support for interchar	109
10.3	Layout	111
10.4	8-bit TeX	112
10.5	LuaTeX	113
10.6	Southeast Asian scripts	120
10.7	CJK line breaking	121
10.8	Arabic justification	123
10.9	Common stuff	127
10.10	Automatic fonts and ids switching	128
10.11	Bidi	134
10.12	Layout	137
10.13	Lua: transforms	146
10.14	Lua: Auto bidi with basic and basic-r	156
11	Data for CJK	168
12	The ‘nil’ language	168
13	Calendars	169
13.1	Islamic	169
13.2	Hebrew	171
13.3	Persian	175
13.4	Coptic and Ethiopic	175
13.5	Buddhist	176
14	Support for Plain T_EX (plain.def)	177
14.1	Not renaming hyphen.tex	177
14.2	Emulating some L ^A T _E X features	178
14.3	General tools	178
14.4	Encoding related macros	182
15	Acknowledgements	185

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

1. Identification and loading of required files

The babel package after unpacking consists of the following files:

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

babel.def is loaded by Plain.

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

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

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

There some additional tex, def and lua files.

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

2. locale directory

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

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

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

3. Tools

```
1 <<version=25.13.102165>>
2 <<date=2025/10/17>>
```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

101 \global\let\bbl@done\empty

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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


```

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

```

3.1. A few core definitions

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

```

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

```

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

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

```

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

```

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

3.3. base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets `ver@babel.sty` so that \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{% Removes trailing dot
315   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
316 \def\bbl@tempe#1=#2\@@{%
317   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
318 \def\bbl@tempd#1.#2\@nnil{%
319   \ifx\@empty#2%
320     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
321   \else
322     \in@{,provide=}{, #1}%
323     \ifin@
324       \edef\bbl@tempc{%
325         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
326     \else
327       \in@{ $modifiers$ }{ $ #1 $ }%
328       \ifin@
329         \bbl@tempe#2\@@
330       \else
331         \in@{ = }{ #1 }%
332         \ifin@
333           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
334         \else
335           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
336           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
337         \fi
338       \fi
339     \fi
340   \fi}
341 \let\bbl@tempc\@empty
342 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
343 \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

```

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

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

```

344 \DeclareOption{KeepShorthandsActive}{}
345 \DeclareOption{activeacute}{}
346 \DeclareOption{activegrave}{}
347 \DeclareOption{debug}{}
348 \DeclareOption{noconfigs}{}
349 \DeclareOption{showlanguages}{}
350 \DeclareOption{silent}{}
351 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
352 \chardef\bbl@iniflag\z@
353 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
354 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
355 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
356 \chardef\bbl@ldfflag\z@
357 \DeclareOption{provide=!}{\chardef\bbl@ldfflag\@ne} % main = 1
358 \DeclareOption{provide+=!}{\chardef\bbl@ldfflag\tw@} % second = 2
359 \DeclareOption{provide*=!}{\chardef\bbl@ldfflag\thr@@} % second + main
360 % Don't use. Experimental.
361 \newif\ifbbl@single
362 \DeclareOption{selectors=off}{\bbl@singletrue}
363 <@More package options>

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

384 \ProcessOptions*

```

3.5. Post-process some options

```

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

```

```

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

```

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

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

```

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

```

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

```

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

```

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

```

416 \edef\bbl@opt@shorthands{%
417 \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

433 \bbl@trace{Defining IfBabelLayout}

```

```

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

```

3.6. Plain: babel.def (start)

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

First, exit immediately if previously loaded.

```

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

```

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

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

```

462 \ifx\ldf@quit\@undefined\else
463 \def\bbl@version{<@version@>}
464 \def\bbl@date{<@date@>}
465 <@Define core switching macros@>

```

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

```

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

```

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

The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s an attempt to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note `l@` is encapsulated, so that its case does not change.

```

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

```

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

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

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

```

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

```

```

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

```

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

```

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

```

4.1. Selecting the language

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

```

547 \let\bbl@select@type\z@
548 \edef\selectlanguage{%
549 \noexpand\protect
550 \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

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

```

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

```

552 \let\xstring\string

```

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

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

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

```

553 \def\bbl@language@stack{}

```

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

\bbl@push@language

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

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

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

```

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

```

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

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

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

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

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

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

```

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

```

```

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

```

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

```

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

```

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

```

742 \long\def\endotherlanguage{\@ignoretrue\ignorespaces}

```

otherlanguage* It is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as ‘figure’. It makes use of `\foreign@language`.

```

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

```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```

750 \expandafter\let\csname endotherlanguage*\endcsname\relax

```

\foreignlanguage This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.

Unlike `\selectlanguage` this command doesn’t switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras{language}` command doesn’t make any global changes. The coding is very similar to part of `\selectlanguage`.

`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a ‘text’ command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

`(3.11) \foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in vmode and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine `\BabelText` which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into hmode with the surrounding lang, and with `\foreignlanguage*` with the new lang.

```

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

```

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

```

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

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

```

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

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

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

```

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

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

```

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

```

\providehyphenmins The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\(language)hyphenmins` is already defined this command has no effect.

```

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

```

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

```

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

```

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

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

```

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\languagegetext\setlocale
```

4.2. Errors

\@nolanerr

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

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

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

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

```
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
907 \ifx\bbl@onlyswitch\empty\endinput\fi
```

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

4.3. More on selection

\babelensure The user command just parses the optional argument and creates a new macro named \bbl@e@<language>. We register a hook at the afterextras event which just executes this macro in a

“complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

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

```

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

```

```

964     \expandafter\bbl@tempa
965     \fi}%
966     \bbl@tempa#1\@empty}
967 \def\bbl@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 \bbl@trace{Short tags}
973 \newcommand\babeltags[1]{%
974   \edef\bbl@tempa{\zap@space#1 \@empty}%
975   \def\bbl@tempb##1=##2\@{%
976     \edef\bbl@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     \bbl@tempc}%
985   \bbl@for\bbl@tempa\bbl@tempa{%
986     \expandafter\bbl@tempb\bbl@tempa\@{}}

```

4.5. Compatibility with language.def

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

```

987 \bbl@trace{Compatibility with language.def}
988 \ifx\directlua\@undefined\else
989   \ifx\bbl@luapatterns\@undefined
990     \input luababel.def
991   \fi
992 \fi
993 \ifx\bbl@languages\@undefined
994   \ifx\directlua\@undefined
995     \openin1 = language.def
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
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@iifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{%
1028     \def\bbl@tempa##1,##3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1029     \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
1030     \bbl@iifunset{\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@#3}}%
1046         \bbl@cs{ev@#2@#3}%
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   \@ifpackagewith{babel}{ensureinfo=off}}}%
1077   {\ifx\InputIfFileExists\@undefined\else
1078     \bbl@ifunset\bbl@lname@#1}%
1079     {\let\bbl@ensuring\@empty % Flag used in babel-serbianc.tex
1080      \def\languagename{#1}%
1081      \bbl@id@assign
1082      \bbl@load@info{#1}}}%
1083   {}%
1084   \fi}%
1085   \expandafter\if\expandafter\@backslashchar
1086     \expandafter\@car\string#2\@nil
1087   \ifx#2\@undefined\else
1088     \ldf@quit{#1}%
1089   \fi
1090 \else
1091   \expandafter\ifx\csname#2\endcsname\relax\else
1092     \ldf@quit{#1}%
1093   \fi
1094 \fi
1095 \bbl@ldfinit}
```

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

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

```

1099 \catcode`\==\eqcatcode \let\eqcatcode\relax
1100 \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.

```

1101 \def\bbl@afterldf{%
1102 \bbl@afterlang
1103 \let\bbl@afterlang\relax
1104 \let\BabelModifiers\relax
1105 \let\bbl@screset\relax}%
1106 \def\ldf@finish#1{%
1107 \loadlocalcfg{#1}%
1108 \bbl@afterldf
1109 \expandafter\main@language\expandafter{#1}%
1110 \catcode`\@=\atcatcode \let\atcatcode\relax
1111 \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*.

```

1112 \@onlypreamble\LdfInit
1113 \@onlypreamble\ldf@quit
1114 \@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.

```

1115 \def\main@language#1{%
1116 \def\bbl@main@language{#1}%
1117 \let\language\name\bbl@main@language
1118 \let\localname\bbl@main@language
1119 \let\mainlocalname\bbl@main@language
1120 \bbl@id@assign
1121 \bbl@patterns{\language}}

```

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

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

```

1122 \def\bbl@beforestart{%
1123 \def\@nolanerr##1{%
1124 \bbl@carg\chardef{l@##1}\z@
1125 \bbl@warning{Undefined language '##1' in aux.\@Reported}}%
1126 \bbl@usehooks{beforestart}{}%
1127 \global\let\bbl@beforestart\relax}
1128 \AtBeginDocument{%
1129 {\@nameuse\bbl@beforestart}}% Group!
1130 \if@files
1131 \providecommand\babel@aux[2]{}%
1132 \immediate\write\@mainaux{\unexpanded{%
1133 \providecommand\babel@aux[2]{\global\let\babel@toc\@gobbletwo}}}%
1134 \immediate\write\@mainaux{\string\@nameuse\bbl@beforestart}}%
1135 \fi
1136 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1137 \ifbbl@single % must go after the line above.
1138 \renewcommand\selectlanguage[1]{}%
1139 \renewcommand\foreignlanguage[2]{#2}%
1140 \global\let\babel@aux\@gobbletwo % Also as flag
1141 \fi}

```

```

1142 %
1143 \ifcase\bbl@engine\or
1144   \AtBeginDocument{\pagedir\bodydir}
1145 \fi

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

1146 \def\select@language@x#1{%
1147   \ifcase\bbl@select@type
1148     \bbl@ifsamestring\language#1\{\select@language{#1}}%
1149   \else
1150     \select@language{#1}%
1151 \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.

```

1152 \bbl@trace{Shorhands}
1153 \def\bbl@withactive#1#2{%
1154   \begingroup
1155     \lccode`~=#2\relax
1156     \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.

```

1157 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1158   \bbl@add\dospecials{do#1}% test \@sanitize = \relax, for back. compat.
1159   \bbl@ifunset{@sanitize}\{\bbl@add\@sanitize{\makeother#1}}%
1160   \ifx\nfss@catcodes\undefined\else
1161     \begingroup
1162       \catcode`#1\active
1163       \nfss@catcodes
1164       \ifnum\catcode`#1=\active
1165         \endgroup
1166         \bbl@add\nfss@catcodes{\makeother#1}%
1167       \else
1168         \endgroup
1169     \fi
1170 \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).

```

1171 \def\bbl@active@def#1#2#3#4{%
1172   \@namedef{#3#1}{%
1173     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1174       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1175     \else
1176       \bbl@afterfi\csname#2@sh@#1@\endcsname
1177     \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.

```

1178   \long\@namedef{#3@arg#1}##1{%
1179     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1180       \bbl@afterelse\csname#4#1\endcsname##1%
1181     \else
1182       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1183     \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.

```

1184 \def\initiate@active@char#1{%
1185   \bbl@ifunset{active@char\string#1}%
1186   {\bbl@withactive
1187     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1188   {}}

```

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).

```

1189 \def\@initiate@active@char#1#2#3{%
1190   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1191   \ifx#1\@undefined
1192     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1193   \else
1194     \bbl@csarg\let{oridef@#2}#1%
1195     \bbl@csarg\edef{oridef@#2}{%
1196       \let\noexpand#1%
1197       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1198   \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*").

```

1199   \ifx#1#3\relax
1200     \expandafter\let\csname normal@char#2\endcsname#3%
1201   \else
1202     \bbl@info{Making #2 an active character}%
1203     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1204     \@namedef{normal@char#2}{%
1205       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
1206   \else
1207     \@namedef{normal@char#2}{#3}%
1208   \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).

```

1209   \bbl@restoreactive{#2}%
1210   \AtBeginDocument{%

```

```

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

```

1218 \let\bbbl@tempa\@firstoftwo
1219 \if\string^#2%
1220 \def\bbbl@tempa{\noexpand\textormath}%
1221 \else
1222 \ifx\bbbl@mathnormal\undefined\else
1223 \let\bbbl@tempa\bbbl@mathnormal
1224 \fi
1225 \fi
1226 \expandafter\edef\csname active@char#2\endcsname{%
1227 \bbbl@tempa
1228 {\noexpand\if@safe@actives
1229 \noexpand\expandafter
1230 \expandafter\noexpand\csname normal@char#2\endcsname
1231 \noexpand\else
1232 \noexpand\expandafter
1233 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1234 \noexpand\fi}%
1235 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1236 \bbbl@csarg\edef{doactive#2}{%
1237 \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

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

(where `\active@char<char>` is *one* control sequence!).

```

1238 \bbbl@csarg\edef{active@#2}{%
1239 \noexpand\active@prefix\noexpand#1%
1240 \expandafter\noexpand\csname active@char#2\endcsname}%
1241 \bbbl@csarg\edef{normal@#2}{%
1242 \noexpand\active@prefix\noexpand#1%
1243 \expandafter\noexpand\csname normal@char#2\endcsname}%
1244 \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.

```

1245 \bbbl@active@def#2\user@group{user@active}{language@active}%
1246 \bbbl@active@def#2\language@group{language@active}{system@active}%
1247 \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.

```

1248 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1249 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1250 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1251 {\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.

```
1252 \if\string'#2%
1253   \let\prim@s\bbl@prim@s
1254   \let\active@math@prime#1%
1255 \fi
1256 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}
```

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

```
1257 << *More package options >> ≡
1258 \DeclareOption{math=active}{}
1259 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1260 << /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*.

```
1261 \ifpackagewith{babel}{KeepShorthandsActive}%
1262   {\let\bbl@restoreactive\@gobble}%
1263   {\def\bbl@restoreactive#1{%
1264     \bbl@exp{%
1265       \\\AfterBabelLanguage\\CurrentOption
1266       {\catcode`#1=\the\catcode`#1\relax}%
1267       \\\AtEndOfPackage
1268       {\catcode`#1=\the\catcode`#1\relax}}}%
1269   \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.

```
1270 \def\bbl@sh@select#1#2{%
1271   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1272     \bbl@afterelse\bbl@scndcs
1273   \else
1274     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1275   \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.

```
1276 \begingroup
1277 \bbl@ifunset{ifincsname}
1278   {\gdef\active@prefix#1{%
1279     \ifx\protect\@typeset@protect
1280     \else
1281       \ifx\protect\@unexpandable@protect
1282         \noexpand#1%
1283       \else
1284         \protect#1%
1285       \fi
1286       \expandafter\@gobble
1287     \fi}}
1288   {\gdef\active@prefix#1{%
1289     \ifincsname
```

```

1290     \string#1%
1291     \expandafter\@gobble
1292   \else
1293     \ifx\protect\@typeset@protect
1294     \else
1295       \ifx\protect\@unexpandable@protect
1296       \noexpand#1%
1297     \else
1298       \protect#1%
1299     \fi
1300     \expandafter\expandafter\expandafter\@gobble
1301   \fi
1302 \fi}}
1303 \endgroup

```

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

```

1304 \newif\if@safe@actives
1305 \@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.

```

1306 \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`.

```

1307 \chardef\bbl@activated\z@
1308 \def\bbl@activate#1{%
1309   \chardef\bbl@activated\@ne
1310   \bbl@withactive{\expandafter\let\expandafter}#1%
1311   \csname bbl@active@\string#1\endcsname}
1312 \def\bbl@deactivate#1{%
1313   \chardef\bbl@activated\tw@
1314   \bbl@withactive{\expandafter\let\expandafter}#1%
1315   \csname bbl@normal@\string#1\endcsname}

```

\bbl@firstcs

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

```

1316 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1317 \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.

```

1318 \def\babel@texpdf#1#2#3#4{%

```

```

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

```

1352 \def\textormath{%
1353   \ifmmode
1354     \expandafter\@secondoftwo
1355   \else
1356     \expandafter\@firstoftwo
1357   \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’.

```

1358 \def\user@group{user}
1359 \def\language@group{english}
1360 \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.

```

1361 \def\useshorthands{%
1362   \@ifstar\bbl@usesh@s{\bbl@usesh@x}}
1363 \def\bbl@usesh@s#1{%
1364   \bbl@usesh@x
1365   {\AddBabelHook{babel-sh-\string#1}{afterextras}}{\bbl@activate{#1}}}%
1366   {#1}}

```

```

1367 \def\bbl@usesh@x#1#2{%
1368   \bbl@ifshorthand{#2}%
1369   {\def\user@group{user}%
1370    \initiate@active@char{#2}%
1371    #1%
1372    \bbl@activate{#2}}%
1373   {\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.

```

1374 \def\user@language@group{user@\language@group}
1375 \def\bbl@set@user@generic#1#2{%
1376   \bbl@ifunset{user@generic@active#1}%
1377   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1378    \bbl@active@def#1\user@group{user@generic@active}{\language@active}%
1379    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1380      \expandafter\noexpand\csname normal@char#1\endcsname}%
1381      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1382        \expandafter\noexpand\csname user@active#1\endcsname}%
1383      \@empty}
1384   \newcommand\defineshorthand[3][user]{%
1385     \edef\bbl@tempa{\zap@space#1 \@empty}%
1386     \bbl@for\bbl@tempb\bbl@tempa{%
1387       \if*\expandafter\@car\bbl@tempb\@nil
1388         \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1389         \@expandtwoargs
1390         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1391       \fi
1392       \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.

```

1393 \def\languageshorthands#1{%
1394   \bbl@ifsamestring{none}{#1}{}%
1395   \bbl@once{short-\localename-#1}{%
1396     \bbl@info{'\localename' activates '#1' shorthands.\Reported}}}%
1397   \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`.

```

1398 \def\aliasshorthand#1#2{%
1399   \bbl@ifshorthand{#2}%
1400   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1401     \ifx\document@notprerr
1402       \@notshorthand{#2}%
1403     \else
1404       \initiate@active@char{#2}%
1405       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1406       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1407       \bbl@activate{#2}%
1408     \fi
1409   \fi}%
1410   {\bbl@error{shorthand-is-off}{#2}{}}}

```

\@notshorthand

```

1411 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}

```

\shorthandon

\shorthandoff The first level definition of these macros just passes the argument on to `\bbl@switch@sh`, adding `\@nil` at the end to denote the end of the list of characters.

```
1412 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1413 \DeclareRobustCommand*\shorthandoff{%
1414   \ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1415 \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.

```
1416 \def\bbl@switch@sh#1#2{%
1417   \ifx#2\@nnil\else
1418     \bbl@ifunset{\bbl@active@\string#2}%
1419     {\bbl@error{not-a-shorthand-b}{\string#2}}}%
1420     {\ifcase#1%   off, on, off*
1421       \catcode`\#2\relax
1422       \or
1423       \catcode`\#2\active
1424       \bbl@ifunset{\bbl@shdef@\string#2}%
1425       {}%
1426       {\bbl@withactive{\expandafter\let\expandafter}\#2%
1427         \csname bbl@shdef@\string#2\endcsname
1428         \bbl@csarg\let{shdef@\string#2}\relax}%
1429       \ifcase\bbl@activated\or
1430         \bbl@activate{\#2}%
1431       \else
1432         \bbl@deactivate{\#2}%
1433       \fi
1434       \or
1435       \bbl@ifunset{\bbl@shdef@\string#2}%
1436       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}\#2}%
1437       {}%
1438       \csname bbl@oricat@\string#2\endcsname
1439       \csname bbl@oridef@\string#2\endcsname
1440       \fi}%
1441   \bbl@afterfi\bbl@switch@sh#1%
1442 \fi}
```

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

```
1443 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1444 \def\bbl@putsh#1{%
1445   \bbl@ifunset{\bbl@active@\string#1}%
1446   {\bbl@putsh@i#1\@empty\@nnil}%
1447   {\csname bbl@active@\string#1\endcsname}}
1448 \def\bbl@putsh@i#1#2\@nnil{%
1449   \csname\language@group @sh@\string#1@%
1450     \ifx\@empty#2\else\string#2@\fi\endcsname}
1451 %
1452 \ifx\bbl@opt@shorthands\@nnil\else
1453   \let\bbl@s@initiate@active@char\initiate@active@char
1454   \def\initiate@active@char#1{%
1455     \bbl@ifshorthand{\#1}{\bbl@s@initiate@active@char{\#1}}{}}
1456   \let\bbl@s@switch@sh\bbl@switch@sh
1457   \def\bbl@switch@sh#1#2{%
1458     \ifx#2\@nnil\else
```

```

1459 \bbl@afterfi
1460 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1461 \fi}
1462 \let\bbl@s@activate\bbl@activate
1463 \def\bbl@activate#1{%
1464 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1465 \let\bbl@s@deactivate\bbl@deactivate
1466 \def\bbl@deactivate#1{%
1467 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1468 \fi

```

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

```

1469 \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.

```

1470 \def\bbl@prim@s{%
1471 \prime\futurelet\@let@token\bbl@pr@m@s}
1472 \def\bbl@if@primes#1#2{%
1473 \ifx#1\@let@token
1474 \expandafter\@firstoftwo
1475 \else\ifx#2\@let@token
1476 \bbl@afterelse\expandafter\@firstoftwo
1477 \else
1478 \bbl@afterfi\expandafter\@secondoftwo
1479 \fi\fi}
1480 \begingroup
1481 \catcode`\^=7 \catcode`\*=\active \lccode`\*='\^
1482 \catcode`\'=12 \catcode`\"=\active \lccode`\"='\ '
1483 \lowercase{%
1484 \gdef\bbl@pr@m@s{%
1485 \bbl@if@primes" '%
1486 \pr@@@s
1487 {\bbl@if@primes*^\pr@@@t\egroup}}}
1488 \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).

```

1489 \initiate@active@char{~}
1490 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1491 \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.

```

1492 \expandafter\def\csname OT1dqpos\endcsname{127}
1493 \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

```

1494 \ifx\f@encoding\undefined
1495 \def\f@encoding{OT1}
1496 \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.

```
1497 \bbl@trace{Language attributes}
1498 \newcommand\languageattribute[2]{%
1499   \def\bbl@tempc{#1}%
1500   \bbl@fixname\bbl@tempc
1501   \bbl@iflanguage\bbl@tempc{%
1502     \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.

```
1503     \ifx\bbl@known@attrs\undefined
1504       \in@false
1505     \else
1506       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1507     \fi
1508     \ifin@
1509       \bbl@warning{%
1510         You have more than once selected the attribute '##1'\%
1511         for language #1. Reported}%
1512     \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.

```
1513       \bbl@info{Activated '##1' attribute for\%
1514         '\bbl@tempc'. Reported}%
1515       \bbl@exp{%
1516         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1517       \edef\bbl@tempa{\bbl@tempc-##1}%
1518       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1519       {\csname\bbl@tempc @attr##1\endcsname}%
1520       {\@attrerr{\bbl@tempc}{##1}}%
1521     \fi}}
1522 \@onlypreamble\languageattribute
```

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

```
1523 \newcommand*\@attrerr[2]{%
1524   \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}`.

```
1525 \def\bbl@declare@ttribute#1#2#3{%
1526   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1527   \ifin@
1528     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1529   \fi
1530   \bbl@add@list\bbl@attributes{#1-#2}%
1531   \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.

```

1532 \def\bbl@ifattributeset#1#2#3#4{%
1533   \ifx\bbl@known@attrs\@undefined
1534     \in@false
1535   \else
1536     \bbl@xin@{,#1-#2,}\bbl@known@attrs,%
1537   \fi
1538   \ifin@
1539     \bbl@afterelse#3%
1540   \else
1541     \bbl@afterfi#4%
1542   \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.

```

1543 \def\bbl@ifknown@ttrib#1#2{%
1544   \let\bbl@tempa\@secondoftwo
1545   \bbl@loopx\bbl@tempb{#2}%
1546   \expandafter\in@\expandafter{\expandafter\bbl@tempb,}\bbl@tempa,%
1547   \ifin@
1548     \let\bbl@tempa\@firstoftwo
1549   \else
1550     \fi}%
1551   \bbl@tempa}

```

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

```

1552 \def\bbl@clear@ttribs{%
1553   \ifx\bbl@attributes\@undefined\else
1554     \bbl@loopx\bbl@tempa{\bbl@attributes}%
1555     \expandafter\bbl@clear@ttrib\bbl@tempa.%
1556   \let\bbl@attributes\@undefined
1557   \fi}
1558 \def\bbl@clear@ttrib#1-#2.{%
1559   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1560 \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.

```

1561 \bbl@trace{Macros for saving definitions}
1562 \def\babel@beginsave{\babel@savecnt\z@}

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

1563 \newcount\babel@savecnt
1564 \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.

```
1565 \def\babel@save#1{%
1566   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1567   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1568     \expandafter\expandafter,\bbl@savedextras,}%
1569   \expandafter\in@\bbl@tempa
1570   \ifin@%else
1571     \bbl@add\bbl@savedextras{,{#1,}}%
1572     \bbl@carg\let\babel@number\babel@savecnt#1\relax
1573     \toks@{\expandafter{\originalTeX\let#1=}}%
1574     \bbl@exp{%
1575       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1576     \advance\babel@savecnt@ne
1577   \fi}
1578 \def\babel@savevariable#1{%
1579   \toks@{\expandafter{\originalTeX #1=}}%
1580   \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`.

```
1581 \def\bbl@redefine#1{%
1582   \edef\bbl@tempa{\bbl@stripslash#1}%
1583   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1584   \expandafter\def\csname\bbl@tempa\endcsname}
1585 \@onlypreamble\bbl@redefine
```

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

```
1586 \def\bbl@redefine@long#1{%
1587   \edef\bbl@tempa{\bbl@stripslash#1}%
1588   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1589   \long\expandafter\def\csname\bbl@tempa\endcsname}
1590 \@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_`.

```
1591 \def\bbl@redefineroobust#1{%
1592   \edef\bbl@tempa{\bbl@stripslash#1}%
1593   \bbl@ifunset{\bbl@tempa\space}%
1594     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1595       \bbl@exp{\def\\#1{\\protect<\bbl@tempa\space>}}}%
1596     {\bbl@exp{\let<org@\bbl@tempa><\bbl@tempa\space>}}}%
1597   \@namedef{\bbl@tempa\space}}
1598 \@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.

```

1599 \def\bbl@frenchspacing{%
1600   \ifnum\the\sfcode`\.\=@m
1601     \let\bbl@nonfrenchspacing\relax
1602   \else
1603     \frenchspacing
1604     \let\bbl@nonfrenchspacing\nonfrenchspacing
1605   \fi}
1606 \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.

```

1607 \let\bbl@elt\relax
1608 \edef\bbl@fs@chars{%
1609   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1610   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1611   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1612 \def\bbl@pre@fs{%
1613   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1614   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1615 \def\bbl@post@fs{%
1616   \bbl@save@sfcodes
1617   \edef\bbl@tempa{\bbl@cl{frspc}}%
1618   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1619   \if u\bbl@tempa      % do nothing
1620   \else\if n\bbl@tempa % non french
1621     \def\bbl@elt##1##2##3{%
1622       \ifnum\sfcode`##1=##2\relax
1623         \babel@savevariable{\sfcode`##1}%
1624         \sfcode`##1=##3\relax
1625       \fi}%
1626     \bbl@fs@chars
1627   \else\if y\bbl@tempa % french
1628     \def\bbl@elt##1##2##3{%
1629       \ifnum\sfcode`##1=##3\relax
1630         \babel@savevariable{\sfcode`##1}%
1631         \sfcode`##1=##2\relax
1632       \fi}%
1633     \bbl@fs@chars
1634   \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.

```

1635 \bbl@trace{Hyphens}
1636 \@onlypreamble\babelhyphenation
1637 \AtEndOfPackage{%
1638   \newcommand\babelhyphenation[2][\@empty]{%
1639     \ifx\bbl@hyphenation@\relax
1640       \let\bbl@hyphenation@\@empty
1641     \fi
1642     \ifx\bbl@hyphlist\@empty\else
1643       \bbl@warning{%
1644         You must not intermingle \string\selectlanguage\space and\\%
1645         \string\babelhyphenation\space or some exceptions will not\\%
1646         be taken into account. Reported}%
1647     \fi

```

```

1648 \ifx\@empty#1%
1649 \protected@edef\bb@hyphenation@\bb@hyphenation\space#2}%
1650 \else
1651 \bb@vforeach{#1}{%
1652 \def\bb@tempa{##1}%
1653 \bb@fixname\bb@tempa
1654 \bb@iflanguage\bb@tempa{%
1655 \bb@csarg\protected@edef\hyphenation@\bb@tempa}{%
1656 \bb@ifunset\bb@hyphenation@\bb@tempa}%
1657 }%
1658 {\csname \bb@hyphenation@\bb@tempa\endcsname\space}%
1659 #2}}}%
1660 \fi}}

```

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

```

1661 \ifx\NewDocumentCommand\@undefined\else
1662 \NewDocumentCommand\babelhyphenmins{sommo}{%
1663 \IfNoValueTF{#2}%
1664 {\protected@edef\bb@hyphenmins@\set@hyphenmins{#3}{#4}}%
1665 \IfValueT{#5}{%
1666 \protected@edef\bb@hyphenatmin@\hyphenationmin=#5\relax}}%
1667 \IfBooleanT{#1}{%
1668 \leftthyphenmin=#3\relax
1669 \rightthyphenmin=#4\relax
1670 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1671 {\edef\bb@tempb{\zap@space#2 \@empty}%
1672 \bb@for\bb@tempa\bb@tempb{%
1673 \@namedef\bb@hyphenmins@\bb@tempa}{\set@hyphenmins{#3}{#4}}%
1674 \IfValueT{#5}{%
1675 \@namedef\bb@hyphenatmin@\bb@tempa}{\hyphenationmin=#5\relax}}}%
1676 \IfBooleanT{#1}{\bb@error{hyphenmins-args}{}}}%
1677 \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.

```

1678 \def\bb@allowhyphens{\ifvmode\else\nobreak\hskip\zap@space\fi}
1679 \def\bb@t@one{Tl}
1680 \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.

```

1681 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1682 \def\babelhyphen{\active@prefix\babelhyphen\bb@hyphen}
1683 \def\bb@hyphen{%
1684 \@ifstar{\bb@hyphen@i @}{\bb@hyphen@i \@empty}}
1685 \def\bb@hyphen@i#1#2{%
1686 \lowercase{\bb@ifunset\bb@hy@#1#2\@empty}}%
1687 {\csname \bb@lusehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1688 {\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.

```

1689 \def\bb@usehyphen#1{%
1690 \leavevmode

```

```

1691 \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1692 \nobreak\hskip\z@skip}
1693 \def\bbl@usehyphen#1{%
1694 \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1695 \def\bbl@hyphenchar{%
1696 \ifnum\hyphenchar\font=\m@ne
1697 \babe\nullhyphen
1698 \else
1699 \char\hyphenchar\font
1700 \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.

```

1701 \def\bbl@hy@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1702 \def\bbl@hy@@soft{\bbl@usehyphen\discretionary{\bbl@hyphenchar}{}}{}
1703 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1704 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1705 \def\bbl@hy@nobreak{\bbl@usehyphen\mbox{\bbl@hyphenchar}}
1706 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1707 \def\bbl@hy@repeat{%
1708 \bbl@usehyphen{
1709 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1710 \def\bbl@hy@@repeat{%
1711 \bbl@usehyphen{
1712 \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1713 \def\bbl@hy@empty{\hskip\z@skip}
1714 \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.

```

1715 \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.

```

1716 \bbl@trace{Multiencoding strings}
1717 \def\bbl@tglobal#1{\global\let#1#1}

```

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

```

1718 << *More package options[] ≡
1719 \DeclareOption{nocase}{}
1720 <</More package options[]

```

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

```

1721 << *More package options[] ≡
1722 \let\bbl@opt@strings\@nnil % accept strings=value
1723 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1724 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1725 \def\BabelStringsDefault{generic}
1726 <</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.

```

1727 \onlypreamble\StartBabelCommands
1728 \def\StartBabelCommands{%
1729   \begingroup
1730   \@tempcnta="7F
1731   \def\bbl@tempa{%
1732     \ifnum\@tempcnta>"FF\else
1733       \catcode\@tempcnta=11
1734       \advance\@tempcnta\@ne
1735       \expandafter\bbl@tempa
1736     \fi}%
1737   \bbl@tempa
1738   <@Macros local to BabelCommands@>
1739   \def\bbl@provstring##1##2{%
1740     \providecommand##1{##2}%
1741     \bbl@tglobal##1}%
1742   \global\let\bbl@scafter\@empty
1743   \let\StartBabelCommands\bbl@startcmds
1744   \ifx\BabelLanguages\relax
1745     \let\BabelLanguages\CurrentOption
1746   \fi
1747   \begingroup
1748   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1749   \StartBabelCommands}
1750 \def\bbl@startcmds{%
1751   \ifx\bbl@screset\@nnil\else
1752     \bbl@usehooks{stopcommands}{}%
1753   \fi
1754   \endgroup
1755   \begingroup
1756   \@ifstar
1757     {\ifx\bbl@opt@strings\@nnil
1758       \let\bbl@opt@strings\BabelStringsDefault
1759     \fi
1760     \bbl@startcmds@i}%
1761   \bbl@startcmds@i}
1762 \def\bbl@startcmds@i##1##2{%
1763   \edef\bbl@L{\zap@space#1 \@empty}%
1764   \edef\bbl@G{\zap@space#2 \@empty}%
1765   \bbl@startcmds@ii}
1766 \let\bbl@startcommands\StartBabelCommands

```

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

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

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

```

1767 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1768   \let\SetString\@gobbletwo
1769   \let\bbl@stringdef\@gobbletwo
1770   \let\AfterBabelCommands\@gobble
1771   \ifx\@empty#1%
1772     \def\bbl@sc@label{generic}%
1773     \def\bbl@encstring##1##2{%
1774       \ProvideTextCommandDefault##1{##2}%
1775       \bbl@tglobal##1%
1776       \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%

```

```

1777 \let\bbl@sctest\in@true
1778 \else
1779 \let\bbl@sc@charset\space % <- zapped below
1780 \let\bbl@sc@fontenc\space % <- " "
1781 \def\bbl@tempa##1=##2\@nil{%
1782 \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1783 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1784 \def\bbl@tempa##1 ##2{% space -> comma
1785 ##1%
1786 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1787 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1788 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1789 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1790 \def\bbl@encstring##1##2{%
1791 \bbl@foreach\bbl@sc@fontenc{%
1792 \bbl@ifunset{T@####1}%
1793 {}%
1794 {\ProvideTextCommand##1{####1}{##2}%
1795 \bbl@tglobal##1%
1796 \expandafter
1797 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1798 \def\bbl@sctest{%
1799 \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1800 \fi
1801 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1802 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1803 \let\AfterBabelCommands\bbl@aftercmds
1804 \let\SetString\bbl@setstring
1805 \let\bbl@stringdef\bbl@encstring
1806 \else % i.e., strings=value
1807 \bbl@sctest
1808 \ifin@
1809 \let\AfterBabelCommands\bbl@aftercmds
1810 \let\SetString\bbl@setstring
1811 \let\bbl@stringdef\bbl@provstring
1812 \fi\fi\fi
1813 \bbl@scswitch
1814 \ifx\bbl@G\@empty
1815 \def\SetString##1##2{%
1816 \bbl@error{missing-group}{##1}{}}}%
1817 \fi
1818 \ifx\@empty#1%
1819 \bbl@usehooks{defaultcommands}{}%
1820 \else
1821 \@expandtwoargs
1822 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}%
1823 \fi}

```

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

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

```

1824 \def\bbl@forlang#1#2{%
1825 \bbl@for#1\bbl@L{%
1826 \bbl@xin@{, #1,}{,\BabelLanguages,}%
1827 \ifin@#2\relax\fi}}
1828 \def\bbl@scswitch{%
1829 \bbl@forlang\bbl@tempa{%
1830 \ifx\bbl@G\@empty\else

```

```

1831 \ifx\SetString@gobbletwo\else
1832 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1833 \bbl@xin@{\bbl@GL,}{\bbl@screset,}%
1834 \ifin@else
1835 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1836 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1837 \fi
1838 \fi
1839 \fi}}
1840 \AtEndOfPackage{%
1841 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1842 \let\bbl@scswitch\relax}
1843 \@onlypreamble\EndBabelCommands
1844 \def\EndBabelCommands{%
1845 \bbl@usehooks{stopcommands}{}}%
1846 \endgroup
1847 \endgroup
1848 \bbl@scafter}
1849 \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.

```

1850 \def\bbl@setstring#1#2{% e.g., \prefacename{<string>}
1851 \bbl@forlang\bbl@tempa{%
1852 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1853 \bbl@ifunset{\bbl@LC}% e.g., \germanchaptername
1854 {\bbl@exp{%
1855 \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\\#1\<\bbl@LC>}}}%
1856 }%
1857 \def\BabelString{#2}%
1858 \bbl@usehooks{stringprocess}{}}%
1859 \expandafter\bbl@stringdef
1860 \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.

```
1861 \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.

```

1862 << *Macros local to BabelCommands >> ≡
1863 \def\SetStringLoop##1##2{%
1864 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1865 \count@\z@
1866 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1867 \advance\count@\@ne
1868 \toks@\expandafter{\bbl@tempa}%
1869 \bbl@exp{%
1870 \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1871 \count@=\the\count@\relax}}}%
1872 << /Macros local to BabelCommands >>

```

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

```

1873 \def\bbl@aftercmds#1{%
1874 \toks@\expandafter{\bbl@scafter#1}%
1875 \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.

```

1876 <<*Macros local to BabelCommands>> ≡
1877   \newcommand\SetCase[3][]{%
1878     \def\bbl@tempa####1####2{%
1879       \ifx####1\@empty\else
1880         \bbl@carg\bbl@add{extras\CurrentOption}{%
1881           \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1882           \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1883           \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1884           \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1885         \expandafter\bbl@tempa
1886       \fi}%
1887   \bbl@tempa##1\@empty\@empty
1888   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1889 <</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.

```

1890 <<*Macros local to BabelCommands>> ≡
1891   \newcommand\SetHyphenMap[1]{%
1892     \bbl@forlang\bbl@tempa{%
1893       \expandafter\bbl@stringdef
1894       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1895 <</Macros local to BabelCommands>>

```

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

```

1896 \newcommand\BabelLower[2]{% one to one.
1897   \ifnum\lccode#1=#2\else
1898     \babel@savevariable{\lccode#1}%
1899     \lccode#1=#2\relax
1900   \fi}
1901 \newcommand\BabelLowerMM[4]{% many-to-many
1902   \@tempcnta=#1\relax
1903   \@tempcntb=#4\relax
1904   \def\bbl@tempa{%
1905     \ifnum\@tempcnta>#2\else
1906       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1907       \advance\@tempcnta#3\relax
1908       \advance\@tempcntb#3\relax
1909       \expandafter\bbl@tempa
1910     \fi}%
1911   \bbl@tempa}
1912 \newcommand\BabelLowerM0[4]{% many-to-one
1913   \@tempcnta=#1\relax
1914   \def\bbl@tempa{%
1915     \ifnum\@tempcnta>#2\else
1916       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1917       \advance\@tempcnta#3
1918       \expandafter\bbl@tempa
1919     \fi}%
1920   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1921 <<*More package options>> ≡
1922 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1923 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1924 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1925 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1926 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1927 <</More package options>>

```


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

```

1928 \AtEndOfPackage{%
1929   \ifx\bbbl@opt@hyphenmap\@undefined
1930     \bbbl@xin@{,}\bbbl@language@opts}%
1931     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1932   \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.

```

1933 \newcommand\setlocalecaption{%
1934   \ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1935 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1936   \bbbl@trim@def\bbbl@tempa{#2}%
1937   \bbbl@xin@{.template}\bbbl@tempa}%
1938   \ifin@
1939     \bbbl@ini@captions@template{#3}{#1}%
1940   \else
1941     \edef\bbbl@tempd{%
1942       \expandafter\expandafter\expandafter
1943       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1944     \bbbl@xin@
1945       {\expandafter\string\csname #2name\endcsname}%
1946       {\bbbl@tempd}%
1947     \ifin@ % Renew caption
1948       \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}%
1949     \ifin@
1950       \bbbl@exp{%
1951         \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1952         {\\bbbl@scset\<#2name>\<#1#2name>}%
1953         {}}%
1954       \else % Old way converts to new way
1955         \bbbl@ifunset{#1#2name}%
1956         {\bbbl@exp{%
1957           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1958           \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1959           {\def\<#2name>\<#1#2name>}}%
1960           {}}}%
1961       {}%
1962     \fi
1963   \else
1964     \bbbl@xin@{\string\bbbl@scset}\bbbl@tempd}% New
1965     \ifin@ % New way
1966       \bbbl@exp{%
1967         \\bbbl@add\<captions#1>{\\bbbl@scset\<#2name>\<#1#2name>}}%
1968         \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1969         {\\bbbl@scset\<#2name>\<#1#2name>}}%
1970         {}}%
1971       \else % Old way, but defined in the new way
1972         \bbbl@exp{%
1973           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
1974           \\bbbl@ifsamestring{\bbbl@tempa}\bbbl@language}%
1975           {\def\<#2name>\<#1#2name>}}%
1976           {}}%
1977       \fi%
1978     \fi
1979     \@namedef{#1#2name}{#3}%
1980     \toks@ \expandafter\bbbl@captionslist}%
1981     \bbbl@exp{\\in@{\<#2name>}\the\toks@}%
1982     \ifin@ \else
1983       \bbbl@exp{\\bbbl@add\\bbbl@captionslist{\<#2name>}}%

```

```

1984 \bbl@tglobal\bbl@captionslist
1985 \fi
1986 \fi}

```

4.15. Making glyphs available

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

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

```

1987 \bbl@trace{Macros related to glyphs}
1988 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
1989 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
1990 \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.

```

1991 \def\save@sf@q#1{\leavevmode
1992 \begingroup
1993 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1994 \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.

```

1995 \ProvideTextCommand{\quotedblbase}{OT1}{%
1996 \save@sf@q{\set@low@box{\textquotedblright\}}%
1997 \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.

```

1998 \ProvideTextCommandDefault{\quotedblbase}{%
1999 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2000 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2001 \save@sf@q{\set@low@box{\textquoteright\}}%
2002 \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.

```

2003 \ProvideTextCommandDefault{\quotesinglbase}{%
2004 \UseTextSymbol{OT1}{\quotesinglbase}}

```

\guillemetleft

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

```

2005 \ProvideTextCommand{\guillemetleft}{OT1}{%
2006 \ifmmode
2007 \ll
2008 \else
2009 \save@sf@q{\nobreak
2010 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2011 \fi}
2012 \ProvideTextCommand{\guillemetright}{OT1}{%
2013 \ifmmode
2014 \gg
2015 \else
2016 \save@sf@q{\nobreak
2017 \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%

```

```

2018 \fi}
2019 \ProvideTextCommand{\guillemotleft}{OT1}{%
2020 \ifmmode
2021 \ll
2022 \else
2023 \save@sf@q{\nobreak
2024 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2025 \fi}
2026 \ProvideTextCommand{\guillemotright}{OT1}{%
2027 \ifmmode
2028 \gg
2029 \else
2030 \save@sf@q{\nobreak
2031 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2032 \fi}

```

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

```

2033 \ProvideTextCommandDefault{\guillemetleft}{%
2034 \UseTextSymbol{OT1}{\guillemetleft}}
2035 \ProvideTextCommandDefault{\guillemetright}{%
2036 \UseTextSymbol{OT1}{\guillemetright}}
2037 \ProvideTextCommandDefault{\guillemotleft}{%
2038 \UseTextSymbol{OT1}{\guillemotleft}}
2039 \ProvideTextCommandDefault{\guillemotright}{%
2040 \UseTextSymbol{OT1}{\guillemotright}}

```

\guilsinglleft

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

```

2041 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2042 \ifmmode
2043 <%
2044 \else
2045 \save@sf@q{\nobreak
2046 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2047 \fi}
2048 \ProvideTextCommand{\guilsinglright}{OT1}{%
2049 \ifmmode
2050 >%
2051 \else
2052 \save@sf@q{\nobreak
2053 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2054 \fi}

```

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

```

2055 \ProvideTextCommandDefault{\guilsinglleft}{%
2056 \UseTextSymbol{OT1}{\guilsinglleft}}
2057 \ProvideTextCommandDefault{\guilsinglright}{%
2058 \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.

```

2059 \DeclareTextCommand{\ij}{OT1}{%
2060 i\kern-0.02em\bbl@allowhyphens j}
2061 \DeclareTextCommand{\IJ}{OT1}{%
2062 I\kern-0.02em\bbl@allowhyphens J}
2063 \DeclareTextCommand{\ij}{T1}{\char188}
2064 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```
2065 \ProvideTextCommandDefault{\ij}{%
2066   \UseTextSymbol{OT1}{\ij}}
2067 \ProvideTextCommandDefault{\IJ}{%
2068   \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).

```
2069 \def\crrtic@{\hrule height0.1ex width0.3em}
2070 \def\crttic@{\hrule height0.1ex width0.33em}
2071 \def\ddj@{%
2072   \setbox0\hbox{d}\dimen@=\ht0
2073   \advance\dimen@lex
2074   \dimen@.45\dimen@
2075   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2076   \advance\dimen@ii.5ex
2077   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2078 \def\DDJ@{%
2079   \setbox0\hbox{D}\dimen@=.55\ht0
2080   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2081   \advance\dimen@ii.15ex % correction for the dash position
2082   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2083   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2084   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2085 %
2086 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2087 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2088 \ProvideTextCommandDefault{\dj}{%
2089   \UseTextSymbol{OT1}{\dj}}
2090 \ProvideTextCommandDefault{\DJ}{%
2091   \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.

```
2092 \DeclareTextCommand{\SS}{OT1}{SS}
2093 \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.

```
2094 \ProvideTextCommandDefault{\glq}{%
2095   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2096 \ProvideTextCommand{\grq}{T1}{%
2097   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2098 \ProvideTextCommand{\grq}{TU}{%
2099   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2100 \ProvideTextCommand{\grq}{OT1}{%
2101   \save@sf@q{\kern-.0125em
2102     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2103     \restore@sf@q}}
```

```

2103 \kern.07em\relax}}
2104 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{0T1}\grq}

```

\glqq

\grqq The ‘german’ double quotes.

```

2105 \ProvideTextCommandDefault{\glqq}{%
2106 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

```

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

```

2107 \ProvideTextCommand{\grqq}{T1}{%
2108 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2109 \ProvideTextCommand{\grqq}{TU}{%
2110 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2111 \ProvideTextCommand{\grqq}{0T1}{%
2112 \save@sf@q{\kern-.07em
2113 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2114 \kern.07em\relax}}
2115 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{0T1}\grqq}

```

\flq

\frq The ‘french’ single guillemets.

```

2116 \ProvideTextCommandDefault{\flq}{%
2117 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2118 \ProvideTextCommandDefault{\frq}{%
2119 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

```

\flqq

\frqq The ‘french’ double guillemets.

```

2120 \ProvideTextCommandDefault{\flqq}{%
2121 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2122 \ProvideTextCommandDefault{\frqq}{%
2123 \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).

```

2124 \def\umlauthigh{%
2125 \def\bbl@umlauta##1{\leavevmode\bgroup%
2126 \accent\csname\f@encoding dqpos\endcsname
2127 ##1\bbl@allowhyphens\egroup}%
2128 \let\bbl@umlaute\bbl@umlauta}
2129 \def\umlautlow{%
2130 \def\bbl@umlauta{\protect\lower@umlaut}}
2131 \def\umlautelow{%
2132 \def\bbl@umlaute{\protect\lower@umlaut}}
2133 \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.

```
2134 \expandafter\ifx\csname U@D\endcsname\relax
2135   \csname newdimen\endcsname\U@D
2136 \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.

```
2137 \def\lower@umlaut#1{%
2138   \leavevmode\bgroup
2139     \U@D lex%
2140     {\setbox\z@\hbox{%
2141       \char\csname f@encoding dqpos\endcsname}%
2142       \dimen@ -.45ex\advance\dimen@ \ht\z@
2143       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2144     \accent\csname f@encoding dqpos\endcsname
2145     \fontdimen5\font\U@D #1%
2146   \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).

```
2147 \AtBeginDocument{%
2148   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2149   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2150   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2151   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{\i}}%
2152   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2153   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2154   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2155   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2156   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2157   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2158   \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.

```
2159 \ifx\l@english\@undefined
2160   \chardef\l@english\z@
2161 \fi
2162 % The following is used to cancel rules in ini files (see Amharic).
2163 \ifx\l@unhyphenated\@undefined
2164   \newlanguage\l@unhyphenated
2165 \fi
```

4.16. Layout

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

```
2166 \bbl@trace{Bidi layout}
2167 \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.

```
2168 \bbl@trace{Input engine specific macros}
2169 \ifcase\bbl@engine
2170   \input txtbabel.def
2171 \or
2172   \input luababel.def
2173 \or
2174   \input xebabel.def
2175 \fi
2176 \providecommand\babelfont{\bbl@error{only-lua-xe}{}}{}{}
2177 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}}{}{}
2178 \ifx\babelposthyphenation\undefined
2179   \let\babelposthyphenation\babelprehyphenation
2180   \let\babelpatterns\babelprehyphenation
2181   \let\babelcharproperty\babelprehyphenation
2182 \fi
2183 \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.

```
2184 \begin{package}
2185 \bbl@trace{Creating languages and reading ini files}
2186 \let\bbl@extend@ini@gobble
2187 \newcommand\babelprovide[2][]{%
2188   \let\bbl@savelangname\languagename
2189   \edef\bbl@savelocaleid{\the\localeid}%
2190   % Set name and locale id
2191   \edef\languagename{#2}%
2192   \bbl@id@assign
2193   % Initialize keys
2194   \bbl@vforeach{captions,date,import,main,script,language,%
2195     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2196     mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2197     Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2198     @import}%
2199     {\bbl@csarg\let{KVP@##1}\@nnil}%
2200   \global\let\bbl@release@transforms@empty
2201   \global\let\bbl@release@casing@empty
2202   \let\bbl@calendars@empty
2203   \global\let\bbl@inidata@empty
2204   \global\let\bbl@extend@ini@gobble
2205   \global\let\bbl@included@inis@empty
2206   \gdef\bbl@key@list{;}%
2207   \bbl@ifunset\bbl@passto@#2{%
2208     {\def\bbl@tempa{#1}}%
2209     {\bbl@exp{\def\\bbl@tempa{[\bbl@passto@#2],\unexpanded{#1}}}}%
2210   \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2211     \in@{/}{##1}% With /, (re)sets a value in the ini
2212     \ifin@
2213       \bbl@renewinikey##1\@{##2}%
2214     \else
2215       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2216         \bbl@error{unknown-provide-key}{##1}{}}{}%
2217     \fi
2218     \bbl@csarg\def{KVP@##1}{##2}%
2219   \fi}%
\end{package}
```

```

2220 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2221 \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2222 % == init ==
2223 \ifx\bbl@screset\@undefined
2224 \bbl@ldfinit
2225 \fi
2226 % ==
2227 % If there is no import (last wins), use @import (internal, there
2228 % must be just one). To consider any order (because
2229 % \PassOptionsToLocale).
2230 \ifx\bbl@KVP@import\@nnil
2231 \let\bbl@KVP@import\bbl@KVP@@import
2232 \fi
2233 % == date (as option) ==
2234 % \ifx\bbl@KVP@date\@nnil\else
2235 % \fi
2236 % ==
2237 \let\bbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2238 \ifcase\bbl@howloaded
2239 \let\bbl@lbfkflag\@empty % new
2240 \else
2241 \ifx\bbl@KVP@hyphenrules\@nnil\else
2242 \let\bbl@lbfkflag\@empty
2243 \fi
2244 \ifx\bbl@KVP@import\@nnil\else
2245 \let\bbl@lbfkflag\@empty
2246 \fi
2247 \fi
2248 % == import, captions ==
2249 \ifx\bbl@KVP@import\@nnil\else
2250 \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
2251 {\ifx\bbl@initoload\relax
2252 \begingroup
2253 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2254 \bbl@input@texini{#2}%
2255 \endgroup
2256 \else
2257 \xdef\bbl@KVP@import{\bbl@initoload}%
2258 \fi}%
2259 {}%
2260 \let\bbl@KVP@date\@empty
2261 \fi
2262 \let\bbl@KVP@captions@@\bbl@KVP@captions
2263 \ifx\bbl@KVP@captions\@nnil
2264 \let\bbl@KVP@captions\bbl@KVP@import
2265 \fi
2266 % ==
2267 \ifx\bbl@KVP@transforms\@nnil\else
2268 \bbl@replace\bbl@KVP@transforms{ }{,}%
2269 \fi
2270 % ==
2271 \ifx\bbl@KVP@mapdot\@nnil\else
2272 \def\bbl@tempa{\@empty}%
2273 \ifx\bbl@KVP@mapdot\bbl@tempa\else
2274 \bbl@exp{\gdef\<bbl@map@@.@@\language\>{\[bbl@KVP@mapdot]}}%
2275 \fi
2276 \fi
2277 % Load ini
2278 % -----
2279 \ifcase\bbl@howloaded
2280 \bbl@provide@new{#2}%
2281 \else
2282 \bbl@ifblank{#1}%

```



```

2283     {}% With \bbl@load@basic below
2284     {\bbl@provide@renew{#2}}%
2285 \fi
2286 % Post tasks
2287 % -----
2288 % == subsequent calls after the first provide for a locale ==
2289 \ifx\bbl@inidata\@empty\else
2290     \bbl@extend@ini{#2}%
2291 \fi
2292 % == ensure captions ==
2293 \ifx\bbl@KVP@captions\@nnil\else
2294     \bbl@ifunset{bbl@extracaps@#2}%
2295         {\bbl@exp{\\babelensure[exclude=\\today]{#2}}}%
2296         {\bbl@exp{\\babelensure[exclude=\\today,
2297             include=\[bbl@extracaps@#2]]{#2}}}%
2298     \bbl@ifunset{bbl@ensure@\language}%
2299         {\bbl@exp{%
2300             \\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2301                 \\foreignlanguage{\language}%
2302                 {###1}}}%
2303         }%
2304     \bbl@exp{%
2305         \\bbl@tglobal\<bbl@ensure@\language>%
2306         \\bbl@tglobal\<bbl@ensure@\language\space>}%
2307 \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.

```

2308 \bbl@load@basic{#2}%
2309 % == script, language ==
2310 % Override the values from ini or defines them
2311 \ifx\bbl@KVP@script\@nnil\else
2312     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2313 \fi
2314 \ifx\bbl@KVP@language\@nnil\else
2315     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2316 \fi
2317 \ifcase\bbl@engine\or
2318     \bbl@ifunset{bbl@chrng@\language}{}%
2319     {\directlua{
2320         Babel.set_chranges_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2321 \fi
2322 % == Line breaking: intraspace, intrapenalty ==
2323 % For CJK, East Asian, Southeast Asian, if interspace in ini
2324 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2325     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2326 \fi
2327 \bbl@provide@intraspace
2328 % == Line breaking: justification ==
2329 \ifx\bbl@KVP@justification\@nnil\else
2330     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2331 \fi
2332 \ifx\bbl@KVP@linebreaking\@nnil\else
2333     \bbl@xin@{,\bbl@KVP@linebreaking,%
2334         {,elongated,kashida,cjk,padding,unhyphenated},%
2335     \ifin@
2336         \bbl@csarg\xdef
2337             {lnbrk@\language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2338     \fi
2339 \fi
2340 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2341 \ifin@e\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi

```

```

2342 \ifin@bbl@arabicjust\fi
2343 \bbl@xin@{/p}{/\bbl@cl\lnbrk}}%
2344 \ifin@AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2345 % == Line breaking: hyphenate.other.(locale|script) ==
2346 \ifx\bbl@lbfkflag\@empty
2347   \bbl@ifunset{bbl@hyotl@\language\language}%
2348   {\bbl@csarg\bbl@replace{hyotl@\language\language}{ }{,}%
2349   \bbl@startcommands*\language\language}%
2350   \bbl@csarg\bbl@foreach{hyotl@\language\language}%
2351   \ifcase\bbl@engine
2352   \ifnum##1<257
2353   \SetHyphenMap{\BabelLower{##1}{##1}}%
2354   \fi
2355   \else
2356   \SetHyphenMap{\BabelLower{##1}{##1}}%
2357   \fi}%
2358   \bbl@endcommands}%
2359 \bbl@ifunset{bbl@hyots@\language\language}%
2360 {\bbl@csarg\bbl@replace{hyots@\language\language}{ }{,}%
2361 \bbl@csarg\bbl@foreach{hyots@\language\language}%
2362 \ifcase\bbl@engine
2363 \ifnum##1<257
2364 \global\lccode##1=##1\relax
2365 \fi
2366 \else
2367 \global\lccode##1=##1\relax
2368 \fi}}%
2369 \fi
2370 % == Counters: maparabic ==
2371 % Native digits, if provided in ini (TeX level, xe and lua)
2372 \ifcase\bbl@engine\else
2373   \bbl@ifunset{bbl@dgnat@\language\language}%
2374   {\expandafter\ifx\csname bbl@dgnat@\language\language\endcsname\@empty\else
2375   \expandafter\expandafter\expandafter
2376   \bbl@setdigits\csname bbl@dgnat@\language\language\endcsname
2377   \ifx\bbl@KVP@maparabic\@nnil\else
2378   \ifx\bbl@latinarabic\@undefined
2379   \expandafter\let\expandafter\@arabic
2380   \csname bbl@counter@\language\language\endcsname
2381   \else % i.e., if layout=counters, which redefines \@arabic
2382   \expandafter\let\expandafter\bbl@latinarabic
2383   \csname bbl@counter@\language\language\endcsname
2384   \fi
2385   \fi
2386   \fi}%
2387 \fi
2388 % == Counters: mapdigits ==
2389 % > luababel.def
2390 % == Counters: alph, Alph ==
2391 \ifx\bbl@KVP@alph\@nnil\else
2392   \bbl@exp{%
2393     \\bbl@add<bbl@preextras@\language\language>{%
2394     \\bbl@save\\@alph
2395     \let\\@alph<bbl@cntr@bbl@KVP@alph @\language\language>}}%
2396   \fi
2397 \ifx\bbl@KVP@Alph\@nnil\else
2398   \bbl@exp{%
2399     \\bbl@add<bbl@preextras@\language\language>{%
2400     \\bbl@save\\@Alph
2401     \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\language\language>}}%
2402   \fi
2403 % == Counters: mapdot ==
2404 \ifx\bbl@KVP@mapdot\@nnil\else

```

```

2405 \bbl@foreach\bbl@list@the{%
2406 \bbl@ifunset{the##1}}{%
2407 {{\bbl@ncarg\let\bbl@tempd{the##1}%
2408 \bbl@carg\bbl@sreplace{the##1}{.}{\bbl@map@lbl{.}}%
2409 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2410 \bbl@exp{\gdef<the##1>{{\the##1}}}%
2411 \fi}}%
2412 \edef\bbl@tempb{enumi,enumii,enumiii,enumiv}%
2413 \bbl@foreach\bbl@tempb{%
2414 \bbl@ifunset{label##1}}{%
2415 {{\bbl@ncarg\let\bbl@tempd{label##1}%
2416 \bbl@carg\bbl@sreplace{label##1}{.}{\bbl@map@lbl{.}}%
2417 \expandafter\ifx\csname label##1\endcsname\bbl@tempd\else
2418 \bbl@exp{\gdef<label##1>{{\label##1}}}%
2419 \fi}}%
2420 \fi
2421 % == Casing ==
2422 \bbl@release@casing
2423 \ifx\bbl@KVP@casing\@nnil\else
2424 \bbl@csarg\xdef{casing@}\languagename}%
2425 {\@nameuse{bbl@casing@}\languagename}\bbl@maybextx\bbl@KVP@casing}%
2426 \fi
2427 % == Calendars ==
2428 \ifx\bbl@KVP@calendar\@nnil
2429 \edef\bbl@KVP@calendar{\bbl@ccl{calpr}}%
2430 \fi
2431 \def\bbl@tempe##1 ##2\@@{% Get first calendar
2432 \def\bbl@tempa{##1}}%
2433 \bbl@exp{\bbl@tempe\bbl@KVP@calendar\space\@@}%
2434 \def\bbl@tempe##1.##2.##3\@@{%
2435 \def\bbl@tempc{##1}%
2436 \def\bbl@tempb{##2}}%
2437 \expandafter\bbl@tempe\bbl@tempa..\@@
2438 \bbl@csarg\edef{calpr@}\languagename}{%
2439 \ifx\bbl@tempc\@empty\else
2440 calendar=\bbl@tempc
2441 \fi
2442 \ifx\bbl@tempb\@empty\else
2443 ,variant=\bbl@tempb
2444 \fi}%
2445 % == engine specific extensions ==
2446 % Defined in XXXbabel.def
2447 \bbl@provide@extra{#2}%
2448 % == require.babel in ini ==
2449 % To load or reload the babel-*.tex, if require.babel in ini
2450 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2451 \bbl@ifunset{bbl@rqtex@}\languagename}{%
2452 {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2453 \let\BabelBeforeIni\@gobbletwo
2454 \chardef\atcatcode=\catcode\@
2455 \catcode\@=11\relax
2456 \def\CurrentOption{#2}%
2457 \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2458 \catcode\@=\atcatcode
2459 \let\atcatcode\relax
2460 \global\bbl@csarg\let{rqtex@\languagename}\relax
2461 \fi}%
2462 \bbl@foreach\bbl@calendars{%
2463 \bbl@ifunset{bbl@ca##1}}{%
2464 \chardef\atcatcode=\catcode\@
2465 \catcode\@=11\relax
2466 \InputIfFileExists{babel-ca-##1.tex}{%
2467 \catcode\@=\atcatcode

```

```

2468     \let\atcatcode\relax}%
2469   {}}%
2470 \fi
2471 % == frenchspacing ==
2472 \ifcase\bbbl@howloaded\in@true\else\in@false\fi
2473 \ifin@else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2474 \ifin@
2475   \bbbl@extras@wrap{\bbbl@pre@fs}%
2476   {\bbbl@pre@fs}%
2477   {\bbbl@post@fs}%
2478 \fi
2479 % == transforms ==
2480 % > luababel.def
2481 \def\CurrentOption{#2}%
2482 \@nameuse{bbbl@icsave@#2}%
2483 % == main ==
2484 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2485   \let\language\bbbl@savelangname
2486   \chardef\localeid\bbbl@savelocaleid\relax
2487 \fi
2488 % == hyphenrules (apply if current) ==
2489 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2490   \ifnum\bbbl@savelocaleid=\localeid
2491     \language\@nameuse{l\language}%
2492   \fi
2493 \fi}

```

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

```

2494 \def\bbbl@provide@new#1{%
2495   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2496   \@namedef{extras#1}{}%
2497   \@namedef{noextras#1}{}%
2498   \bbbl@startcommands*{#1}{captions}%
2499   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2500     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2501       \ifx##1\@nnil\else
2502         \bbbl@exp{%
2503           \\SetString\\##1{%
2504             \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2505           \expandafter\bbbl@tempb
2506         \fi}%
2507     \expandafter\bbbl@tempb\bbbl@captionslist\@nnil
2508   \else
2509     \ifx\bbbl@initoload\relax
2510       \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2511     \else
2512       \bbbl@read@ini{\bbbl@initoload}2% % Same
2513     \fi
2514   \fi
2515   \StartBabelCommands*{#1}{date}%
2516   \ifx\bbbl@KVP@date\@nnil
2517     \bbbl@exp{%
2518       \\SetString\\today{\\bbbl@nocaption{today}{#1today}}}%
2519   \else
2520     \bbbl@savetoday
2521     \bbbl@savedate
2522   \fi
2523   \bbbl@endcommands
2524   \bbbl@load@basic{#1}%
2525   % == hyphenmins == (only if new)
2526   \bbbl@exp{%
2527     \gdef\<#1hyphenmins>{%

```

```

2528      {\bbl@ifunset{\bbl@lftm@#1}{2}{\bbl@cs{lftm@#1}}}%
2529      {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
2530 % == hyphenrules (also in renew) ==
2531 \bbl@provide@hyphens{#1}%
2532 % == main ==
2533 \ifx\bbl@KVP@main\@nnil\else
2534   \expandafter\main@language\expandafter{#1}%
2535 \fi}
2536 %
2537 \def\bbl@provide@renew#1{%
2538   \ifx\bbl@KVP@captions\@nnil\else
2539     \StartBabelCommands*{#1}{captions}%
2540     \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2541   \EndBabelCommands
2542 \fi
2543 \ifx\bbl@KVP@date\@nnil\else
2544   \StartBabelCommands*{#1}{date}%
2545   \bbl@savetoday
2546   \bbl@savestate
2547   \EndBabelCommands
2548 \fi
2549 % == hyphenrules (also in new) ==
2550 \ifx\bbl@lbkflag\@empty
2551   \bbl@provide@hyphens{#1}%
2552 \fi
2553 % == main ==
2554 \ifx\bbl@KVP@main\@nnil\else
2555   \expandafter\main@language\expandafter{#1}%
2556 \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.

```

2557 \def\bbl@load@basic#1{%
2558   \ifcase\bbl@howloaded\or\or
2559     \ifcase\csname bbl@llevel@\language\endcsname
2560       \bbl@csarg\let\lname@\language\relax
2561     \fi
2562   \fi
2563   \bbl@ifunset{\bbl@lname@#1}%
2564   {\def\BabelBeforeIni##1##2{%
2565     \begingroup
2566       \let\bbl@ini@captions@aux\@gobbletwo
2567       \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2568       \bbl@read@ini{##1}1%
2569       \ifx\bbl@initoload\relax\endinput\fi
2570     \endgroup}%
2571   \begingroup % boxed, to avoid extra spaces:
2572     \ifx\bbl@initoload\relax
2573       \bbl@input@texini{#1}%
2574     \else
2575       \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2576     \fi
2577   \endgroup}%
2578   {}}

```

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).

```

2579 \def\bbl@load@info#1{%
2580   \def\BabelBeforeIni##1##2{%
2581     \begingroup
2582       \bbl@read@ini{##1}0%

```

```

2583 \endinput % babel- .tex may contain onlypreamble's
2584 \endgroup}% boxed, to avoid extra spaces:
2585 {\bbl@input@texini{#1}}

```

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

```

2586 \def\bbl@provide@hyphens#1{%
2587 \@tempcnta\m@ne % a flag
2588 \ifx\bbl@KVP@hyphenrules\@nnil\else
2589 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2590 \bbl@foreach\bbl@KVP@hyphenrules{%
2591 \ifnum\@tempcnta=\m@ne % if not yet found
2592 \bbl@ifsamestring{##1}{+}%
2593 {\bbl@carg\addlanguage{l@##1}}%
2594 }%
2595 \bbl@ifunset{l@##1}% After a possible +
2596 }%
2597 {\@tempcnta\@nameuse{l@##1}}%
2598 \fi}%
2599 \ifnum\@tempcnta=\m@ne
2600 \bbl@warning{%
2601 Requested 'hyphenrules' for '\language' not found:\\%
2602 \bbl@KVP@hyphenrules.\\%
2603 Using the default value. Reported}%
2604 \fi
2605 \fi
2606 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2607 \ifx\bbl@KVP@captions@\@nnil
2608 \bbl@ifunset\bbl@hyphr{#1}{}% use value in ini, if exists
2609 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2610 }%
2611 {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2612 }% if hyphenrules found:
2613 {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}%
2614 \fi
2615 \fi
2616 \bbl@ifunset{l@#1}%
2617 {\ifnum\@tempcnta=\m@ne
2618 \bbl@carg\adddialect{l@#1}\language
2619 \else
2620 \bbl@carg\adddialect{l@#1}\@tempcnta
2621 \fi}%
2622 {\ifnum\@tempcnta=\m@ne\else
2623 \global\bbl@carg\chardef{l@#1}\@tempcnta
2624 \fi}}

```

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

```

2625 \def\bbl@input@texini#1{%
2626 \bbl@bsphack
2627 \bbl@exp{%
2628 \catcode`\\%=14 \catcode`\\%=0
2629 \catcode`\\%=1 \catcode`\\%=2
2630 \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2631 \catcode`\\%=the\catcode`%\relax
2632 \catcode`\\%=the\catcode`\\relax
2633 \catcode`\\%=the\catcode`\\relax
2634 \catcode`\\%=the\catcode`\\relax}%
2635 \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.

```

2636 \def\bbl@iniline#1\bbl@iniline{%

```

```

2637 \ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2638 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2639 \def\bbl@iniskip#1\@@{\% if starts with ;
2640 \def\bbl@inistore#1=#2\@@{\% full (default)
2641 \bbl@trim\def\bbl@tempa{#1}%
2642 \bbl@trim\toks@{#2}%
2643 \bbl@ifsamestring{\bbl@tempa}{\include}%
2644 {\bbl@read@subini{\the\toks@}}%
2645 {\bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2646 \ifin@
2647 \bbl@xin@{,identification/include.}%
2648 {,\bbl@section/\bbl@tempa}%
2649 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2650 \bbl@exp{%
2651 \\\g@addto@macro\\bbl@inidata{%
2652 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2653 \fi}}
2654 \def\bbl@inistore@min#1=#2\@@{\% minimal (maybe set in \bbl@read@ini)
2655 \bbl@trim\def\bbl@tempa{#1}%
2656 \bbl@trim\toks@{#2}%
2657 \bbl@xin@{.identification.}{.\bbl@section.}%
2658 \ifin@
2659 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2660 \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2661 \fi}

```

4.19. Main loop in ‘provide’

Now, the ‘main loop’, `\bbl@read@ini`, which **must be executed inside a group**. At this point, `\bbl@inidata` may contain data declared in `\babelprovide`, with ‘slashed’ keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, ‘export’ some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with `\babelprovide` it’s either 1 (without import) or 2 (which import). The value `−1` is used with `\DocumentMetadata`.

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

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

```

2662 \def\bbl@loop@ini#1{%
2663 \loop
2664 \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2665 \endlinechar\m@ne
2666 \read#1 to \bbl@line
2667 \endlinechar\^^M
2668 \ifx\bbl@line\empty\else
2669 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2670 \fi
2671 \repeat}
2672 %
2673 \def\bbl@read@subini#1{%
2674 \ifx\bbl@readsubstream\undefined
2675 \csname newread\endcsname\bbl@readsubstream
2676 \fi
2677 \openin\bbl@readsubstream=babel-#1.ini
2678 \ifeof\bbl@readsubstream
2679 \bbl@error{no-ini-file}{#1}{}%
2680 \else
2681 {\bbl@loop@ini\bbl@readsubstream}%
2682 \fi
2683 \closein\bbl@readsubstream}
2684 %

```

```

2685 \ifx\bbl@readstream\@undefined
2686 \csname newread\endcsname\bbl@readstream
2687 \fi
2688 \def\bbl@read@ini#1#2{%
2689 \global\let\bbl@extend@ini\@gobble
2690 \openin\bbl@readstream=babel-#1.ini
2691 \ifeof\bbl@readstream
2692 \bbl@error{no-ini-file}{#1}{}{}%
2693 \else
2694 % == Store ini data in \bbl@inidata ==
2695 \catcode\ =10 \catcode`\"=12
2696 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2697 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2698 \ifnum#2=\m@ne % Just for the info
2699 \edef\language{tag \bbl@metalang}%
2700 \fi
2701 \bbl@info{\ifnum#2=\m@ne Fetching locale name for tag \bbl@metalang
2702 \else Importing
2703 \ifcase#2font and identification \or basic \fi
2704 data for \language
2705 \fi}%
2706 from babel-#1.ini. Reported}%
2707 \ifnum#2<\@ne
2708 \global\let\bbl@inidata\@empty
2709 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2710 \fi
2711 \def\bbl@section{identification}%
2712 \bbl@exp{%
2713 \\\bbl@inistore tag.ini=#1\\ \@
2714 \\\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\ \@}%
2715 \bbl@loop@ini\bbl@readstream
2716 % == Process stored data ==
2717 \ifnum#2=\m@ne
2718 \def\bbl@tempa##1 ##2\@{##1}% Get first name
2719 \def\bbl@elt##1##2##3{%
2720 \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2721 {\edef\language{\bbl@tempa##3 \@}%
2722 \bbl@id@assign
2723 \def\bbl@elt####1####2####3{}}%
2724 {}}%
2725 \bbl@inidata
2726 \fi
2727 \bbl@csarg\xdef{lini@\language}{#1}%
2728 \bbl@read@ini@aux
2729 % == 'Export' data ==
2730 \bbl@ini@exports{#2}%
2731 \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2732 \global\let\bbl@inidata\@empty
2733 \bbl@exp{\\ \bbl@add@list\\ \bbl@ini@loaded{\language}}%
2734 \bbl@tglobal\bbl@ini@loaded
2735 \fi
2736 \closein\bbl@readstream}
2737 \def\bbl@read@ini@aux{%
2738 \let\bbl@savestrings\@empty
2739 \let\bbl@savetoday\@empty
2740 \let\bbl@savestate\@empty
2741 \def\bbl@elt##1##2##3{%
2742 \def\bbl@section{##1}%
2743 \in@{=date.}{=##1}% Find a better place
2744 \ifin@
2745 \bbl@ifunset{bbl@inikv{##1}}%
2746 {\bbl@ini@calendar{##1}}%
2747 {}}%

```



```

2748 \fi
2749 \bbl@ifunset{bbl@inikv@##1}{}%
2750 {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2751 \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.

```

2752 \def\bbl@extend@ini@aux#1{%
2753 \bbl@startcommands*{#1}{captions}%
2754 % Activate captions/... and modify exports
2755 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2756 \setlocalecaption{#1}{##1}{##2}}}%
2757 \def\bbl@inikv@captions##1##2{%
2758 \bbl@ini@captions@aux{##1}{##2}}}%
2759 \def\bbl@stringdef##1##2{\gdef##1{##2}}}%
2760 \def\bbl@exportkey##1##2##3{%
2761 \bbl@ifunset{bbl@kv@##2}{}%
2762 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2763 \bbl@exp{\global\let\<bbl@##1@language\>\<bbl@kv@##2>}}}%
2764 \fi}}}%
2765 % As with \bbl@read@ini, but with some changes
2766 \bbl@read@ini@aux
2767 \bbl@ini@exports\tw@
2768 % Update inidata@lang by pretending the ini is read.
2769 \def\bbl@elt##1##2##3{%
2770 \def\bbl@section{##1}%
2771 \bbl@iniline##2=##3\bbl@iniline}%
2772 \csname bbl@inidata@#1\endcsname
2773 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2774 \StartBabelCommands*{#1}{date}% And from the import stuff
2775 \def\bbl@stringdef##1##2{\gdef##1{##2}}}%
2776 \bbl@savetoday
2777 \bbl@savestate
2778 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2779 \def\bbl@ini@calendar#1{%
2780 \lowercase{\def\bbl@tempa{=#1=}}}%
2781 \bbl@replace\bbl@tempa{=date.gregorian}{}}}%
2782 \bbl@replace\bbl@tempa{=date.}{}}}%
2783 \in@{.licr=}{#1=}%
2784 \ifin@
2785 \ifcase\bbl@engine
2786 \bbl@replace\bbl@tempa{.licr=}{}}}%
2787 \else
2788 \let\bbl@tempa\relax
2789 \fi
2790 \fi
2791 \ifx\bbl@tempa\relax\else
2792 \bbl@replace\bbl@tempa{=}{}}}%
2793 \ifx\bbl@tempa\@empty\else
2794 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2795 \fi
2796 \bbl@exp{%
2797 \def\bbl@inikv@#1>####1####2{%
2798 \\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2799 \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).

```

2800 \def\bbl@renewinikey#1/#2\@#3{%
2801 \global\let\bbl@extend@ini\bbl@extend@ini@aux

```

```

2802 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2803 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2804 \bbl@trim\toks@{#3}% value
2805 \bbl@exp{%
2806 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2807 \\g@addto@macro\\bbl@inidata{%
2808 \\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.

```

2809 \def\bbl@exportkey#1#2#3{%
2810 \bbl@ifunset{\bbl@kv@#2}%
2811 {\bbl@csarg\gdef{#1@\language\language}\{#3}}%
2812 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
2813 \bbl@csarg\gdef{#1@\language\language}\{#3}}%
2814 \else
2815 \bbl@exp{\global\let<\bbl@#1@\language\language>\<\bbl@kv@#2>}%
2816 \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.

```

2817 \def\bbl@iniwarning#1{%
2818 \bbl@ifunset{\bbl@kv@identification.warning#1}{}}%
2819 {\bbl@warning{%
2820 From babel-\bbl@cs{lini@\language\language}.ini:\\%
2821 \bbl@cs{@kv@identification.warning#1}\\%
2822 Reported}}}
2823 %
2824 \let\bbl@release@transforms\@empty
2825 \let\bbl@release@casing\@empty

```

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

```

2826 \def\bbl@ini@exports#1{%
2827 % Identification always exported
2828 \bbl@iniwarning{}%
2829 \ifcase\bbl@engine
2830 \bbl@iniwarning{.pdf\latex}%
2831 \or
2832 \bbl@iniwarning{.lua\latex}%
2833 \or
2834 \bbl@iniwarning{.xel\latex}%
2835 \fi%
2836 \bbl@exportkey{lllevel}{identification.load.level}{}%
2837 \bbl@exportkey{elname}{identification.name.english}{}%
2838 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2839 {\csname\bbl@elname@\language\language\endcsname}}%
2840 \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2841 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2842 \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2843 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2844 \bbl@exportkey{esname}{identification.script.name}{}%

```

```

2845 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2846   {\csname bbl@esname@language\endcsname}}%
2847 \bbl@exportkey{sbc}{identification.script.tag.bcp47}}}%
2848 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2849 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}}}%
2850 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}}}%
2851 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}}}%
2852 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}}}%
2853 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}}}%
2854 % Also maps bcp47 -> language
2855 \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
2856 \ifcase\bbl@engine\or
2857   \directlua{%
2858     Babel.locale_props[\the\bbl@cs{id@language}].script
2859     = '\bbl@cl{sbc}}}%
2860 \fi
2861 % Conditional
2862 \ifnum#1>z@ % -1 or 0 = only info, 1 = basic, 2 = (re)new
2863 \bbl@exportkey{calpr}{date.calendar.preferred}}}%
2864 \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
2865 \bbl@exportkey{hyphr}{typography.hyphenrules}}}%
2866 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2867 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2868 \bbl@exportkey{prehc}{typography.prehyphenchar}}}%
2869 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}}}%
2870 \bbl@exportkey{hyots}{typography.hyphenate.other.script}}}%
2871 \bbl@exportkey{intsp}{typography.intraspace}}}%
2872 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2873 \bbl@exportkey{chrng}{characters.ranges}}}%
2874 \bbl@exportkey{quote}{characters.delimiters.quotes}}}%
2875 \bbl@exportkey{dgnat}{numbers.digits.native}}}%
2876 \ifnum#1=\tw@ % only (re)new
2877 \bbl@exportkey{rqtex}{identification.require.babel}}}%
2878 \bbl@tglobal\bbl@savetoday
2879 \bbl@tglobal\bbl@savestate
2880 \bbl@savestrings
2881 \fi
2882 \fi}

```

4.20. Processing keys in ini

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

```

2883 \def\bbl@inikv#1#2{%      key=value
2884   \toks@{#2}%             This hides #'s from ini values
2885   \bbl@csarg\xdef{kv@bbl@section.#1}{\the\toks@}}

```

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

```

2886 \let\bbl@inikv@identification\bbl@inikv
2887 \let\bbl@inikv@date\bbl@inikv
2888 \let\bbl@inikv@typography\bbl@inikv
2889 \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.

```

2890 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@language}\empty x-\fi}
2891 \def\bbl@inikv@characters#1#2{%
2892   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2893   {\bbl@exp{%
2894     \\g@addto@macro\\bbl@release@casing{%
2895       \\bbl@casemapping}{\language}{\unexpanded{#2}}}%
2896     {\in@{casing.}{#1}% e.g., casing.Uv = uV
2897     \ifin@

```

```

2898 \lowercase{\def\bbl@tempb{#1}}%
2899 \bbl@replace\bbl@tempb{casing.}{}%
2900 \bbl@exp{\g@addto@macro{\bbl@release@casing}%
2901 \bbl@casemapping
2902 {\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2903 \else
2904 \bbl@inikv{#1}{#2}%
2905 \fi}

```

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

```

2906 \def\bbl@inikv@counters#1#2{%
2907 \bbl@ifsamestring{#1}{digits}%
2908 {\bbl@error{digits-is-reserved}{}}}%
2909 {}%
2910 \def\bbl@tempc{#1}%
2911 \bbl@trim@def{\bbl@tempb*}{#2}%
2912 \in@{.1$}{#1$}%
2913 \ifin@
2914 \bbl@replace\bbl@tempc{.1}{}%
2915 \bbl@csarg\protected@xdef{cnt@#1\bbl@tempc @\language}{%
2916 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2917 \fi
2918 \in@{.F.}{#1}%
2919 \ifin@else\in@{.S.}{#1}\fi
2920 \ifin@
2921 \bbl@csarg\protected@xdef{cnt@#1\bbl@tempb*}{%
2922 \else
2923 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2924 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2925 \bbl@csarg{\global\expandafter\let}{cnt@#1\bbl@tempa}
2926 \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.

```

2927 \ifcase\bbl@engine
2928 \bbl@csarg\def{inikv@captions.licr}#1#2{%
2929 \bbl@ini@captions@aux{#1}{#2}}
2930 \else
2931 \def\bbl@inikv@captions#1#2{%
2932 \bbl@ini@captions@aux{#1}{#2}}
2933 \fi

```

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

```

2934 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2935 \bbl@replace\bbl@tempa{.template}{}%
2936 \def\bbl@toreplace{#1}{}%
2937 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
2938 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
2939 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
2940 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
2941 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
2942 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2943 \ifin@
2944 \@nameuse{bbl@patch\bbl@tempa}%
2945 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2946 \fi
2947 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2948 \ifin@
2949 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2950 \bbl@exp{\gdef<fnum@\bbl@tempa>{%
2951 \bbl@ifunset{bbl@\bbl@tempa fmt@\language}%

```

```

2952      {\fnum@bbl@tempa}}%
2953      {\@nameuse{bbl@bbl@tempa fmt@\\language}}}%
2954 \fi}
2955 %
2956 \def\bbl@ini@captions@aux#1#2{%
2957   \bbl@trim@def\bbl@tempa{#1}%
2958   \bbl@xin@{.template}{\bbl@tempa}%
2959   \ifin@
2960     \bbl@ini@captions@template{#2}\language
2961   \else
2962     \bbl@ifblank{#2}%
2963     {\bbl@exp{%
2964       \toks@{\@nameuse{bbl@nocaption}{\bbl@tempa}{\language\bbl@tempa name}}}%
2965       {\bbl@trim\toks@{#2}}}%
2966     \bbl@exp{%
2967       \@nameuse{bbl@add}\bbl@savestrings{%
2968         \SetString<\bbl@tempa name>{\the\toks@}}%
2969       \toks@expandafter{\bbl@captionslist}%
2970       \bbl@exp{\@nameuse{<\bbl@tempa name>}{\the\toks@}}%
2971       \ifin@else
2972         \bbl@exp{%
2973           \@nameuse{bbl@add}<\bbl@extracaps@language>{\<\bbl@tempa name>}%
2974           \@nameuse{bbl@tglobal}<\bbl@extracaps@language>}%
2975       \fi
2976 \fi}

```

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

```

2977 \def\bbl@list@the{%
2978   part,chapter,section,subsection,subsubsection,paragraph,%
2979   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
2980   table,page,footnote,mpfootnote,mpfn}
2981 %
2982 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
2983   \bbl@ifunset{bbl@map@#1@language}%
2984   {\@nameuse{#1}}%
2985   {\@nameuse{bbl@map@#1@language}}}
2986 %
2987 \def\bbl@map@lbl#1{% #1:a sign, eg, .
2988   \ifincsname#1\else
2989     \bbl@ifunset{bbl@map@#1@language}%
2990     {#1}%
2991     {\@nameuse{bbl@map@#1@language}}%
2992 \fi}
2993 %
2994 \def\bbl@inikv@labels#1#2{%
2995   \in@{.map}{#1}%
2996   \ifin@
2997     \in@{,dot.map,}{, #1,}%
2998   \ifin@
2999     \global\@namedef{bbl@map@.@@@language}{#2}%
3000 \fi
3001 \ifx\bbl@KVP@labels\@nnil\else
3002   \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3003   \ifin@
3004     \def\bbl@tempc{#1}%
3005     \bbl@replace\bbl@tempc{.map}{}%
3006     \in@{, #2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3007     \bbl@exp{%
3008       \gdef<\bbl@map@bbl@tempc @language>%
3009       {\ifin@<#2>\else\\loccounter{#2}\fi}}%
3010     \bbl@foreach\bbl@list@the{%
3011       \bbl@ifunset{the##1}{%
3012         {\bbl@ncarg\let\bbl@tempd{the##1}%

```

```

3013 \bbl@exp{%
3014 \\\bbl@sreplace\<the##1>%
3015 {\<\bbl@tempc>{##1}}%
3016 {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3017 \\\bbl@sreplace\<the##1>%
3018 {\<\@empty @\bbl@tempc>\<c@##1>%
3019 {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3020 \\\bbl@sreplace\<the##1>%
3021 {\\\csname @\bbl@tempc\\endcsname\<c@##1>%
3022 {\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3023 \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3024 \bbl@exp{\gdef\<the##1>{\[the##1]}}%
3025 \fi}%
3026 \fi
3027 \fi
3028 %
3029 \else
3030 % The following code is still under study. You can test it and make
3031 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3032 % language dependent.
3033 \in@{enumerate.}{#1}%
3034 \ifin@
3035 \def\bbl@tempa{#1}%
3036 \bbl@replace\bbl@tempa{enumerate.}{}%
3037 \def\bbl@toreplace{#2}%
3038 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3039 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3040 \bbl@replace\bbl@toreplace{[ ]}{\endcsname{}}%
3041 \toks@{\expandafter\bbl@toreplace}%
3042 \bbl@exp{%
3043 \\\bbl@add\<extras\language>{%
3044 \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3045 \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}%
3046 \\\bbl@toggle\<extras\language>}%
3047 \fi
3048 \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.

```

3049 \def\bbl@chapttype{chapter}
3050 \ifx\@makechapterhead\undefined
3051 \let\bbl@patchchapter\relax
3052 \else\ifx\thechapter\undefined
3053 \let\bbl@patchchapter\relax
3054 \else\ifx\ps@headings\undefined
3055 \let\bbl@patchchapter\relax
3056 \else
3057 \def\bbl@patchchapter{%
3058 \global\let\bbl@patchchapter\relax
3059 \gdef\bbl@chfmt{%
3060 \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3061 {\@chapapp\space\thechapter}%
3062 {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}%
3063 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3064 \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3065 \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3066 \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3067 \bbl@toggle\appendix
3068 \bbl@toggle\ps@headings
3069 \bbl@toggle\chaptermark
3070 \bbl@toggle\@makechapterhead}

```

```

3071 \let\bbl@patchappendix\bbl@patchchapter
3072 \fi\fi\fi
3073 \ifx\@part\@undefined
3074 \let\bbl@patchpart\relax
3075 \else
3076 \def\bbl@patchpart{%
3077   \global\let\bbl@patchpart\relax
3078   \gdef\bbl@partformat{%
3079     \bbl@ifunset\bbl@partfmt@\language\name}%
3080     {\partname\nobreakspace\thepart}%
3081     {\@nameuse\bbl@partfmt@\language\name}}}%
3082   \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3083   \bbl@tglobal\@part}
3084 \fi

```

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

```

3085 \let\bbl@calendar\@empty
3086 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3087 \def\bbl@localedate#1#2#3#4{%
3088   \begingroup
3089     \edef\bbl@they{#2}%
3090     \edef\bbl@them{#3}%
3091     \edef\bbl@thed{#4}%
3092     \edef\bbl@tempe{%
3093       \bbl@ifunset\bbl@calpr@\language\name}{\bbl@cl{calpr}},%
3094       #1}%
3095     \bbl@exp{\lowercase{\edef\\bbl@tempe{\bbl@tempe}}}%
3096     \bbl@replace\bbl@tempe{ }{}%
3097     \bbl@replace\bbl@tempe{convert}{convert=}%
3098     \let\bbl@ld@calendar\@empty
3099     \let\bbl@ld@variant\@empty
3100     \let\bbl@ld@convert\relax
3101     \def\bbl@tempb##1=##2\@{\@namedef\bbl@ld@##1}{##2}}%
3102     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3103     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3104     \ifx\bbl@ld@calendar\@empty\else
3105       \ifx\bbl@ld@convert\relax\else
3106         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3107         {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3108       \fi
3109     \fi
3110     \@nameuse\bbl@precalendar% Remove, e.g., +, -civil (-ca-islamic)
3111     \edef\bbl@calendar{% Used in \month..., too
3112       \bbl@ld@calendar
3113       \ifx\bbl@ld@variant\@empty\else
3114         .\bbl@ld@variant
3115       \fi}%
3116     \bbl@cased
3117     {\@nameuse\bbl@date@\language\name @\bbl@calendar}%
3118     \bbl@they\bbl@them\bbl@thed}%
3119   \endgroup}
3120 %
3121 \def\bbl@printdate#1{%
3122   \@ifnextchar[\bbl@printdate@i{#1}]{\bbl@printdate@i{#1}}{}}
3123 \def\bbl@printdate@i#1[#2]#3#4#5{%
3124   \bbl@usedategrouptrue
3125   \@nameuse\bbl@ensure@#1{\localedate[#2][#3][#4][#5]}%
3126 %
3127 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3128 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3129   \bbl@trim\def\bbl@tempa{#1.#2}%
3130   \bbl@ifsamestring\bbl@tempa{months.wide}% to savedate

```

```

3131 {\bbl@trim@def\bbl@tempa{#3}%
3132 \bbl@trim\toks@{#5}%
3133 \@temptokena\expandafter{\bbl@savestate}%
3134 \bbl@exp{% Reverse order - in ini last wins
3135 \def\\bbl@savestate{%
3136 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3137 \the\@temptokena}}}%
3138 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3139 {\lowercase{\def\bbl@tempb{#6}}}%
3140 \bbl@trim@def\bbl@toreplace{#5}%
3141 \bbl@TG@@date
3142 \global\bbl@csarg\let{date@\language name @\bbl@tempb}\bbl@toreplace
3143 \ifx\bbl@savestate@empty
3144 \bbl@exp{%
3145 \\AfterBabelCommands{%
3146 \gdef\<\language name date>{\\protect\<\language name date >}%
3147 \gdef\<\language name date >{\\bbl@printdate{\language name}}}%
3148 \def\\bbl@savestate{%
3149 \\SetString\\today{%
3150 \<\language name date>[convert]%
3151 {\the\year}{\the\month}{\the\day}}}%
3152 \fi}%
3153 {}}}

```

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).

```

3154 \let\bbl@calendar@empty
3155 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3156 \@nameuse{bbl@ca@#2}#1@@}
3157 \newcommand\babelDateSpace{\nobreakspace}
3158 \newcommand\babelDateDot{. \@}
3159 \newcommand\babelDated[1]{\number#1}
3160 \newcommand\babelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3161 \newcommand\babelDateM[1]{\number#1}
3162 \newcommand\babelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3163 \newcommand\babelDateMMM[1]{%
3164 \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3165 \newcommand\babelDatey[1]{\number#1}%
3166 \newcommand\babelDateyy[1]{%
3167 \ifnum#1<10 0\number#1 %
3168 \else\ifnum#1<100 \number#1 %
3169 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3170 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3171 \else
3172 \bbl@error{limit-two-digits}{\number#1}%
3173 \fi\fi\fi\fi}
3174 \newcommand\babelDateyyyy[1]{\number#1}
3175 \newcommand\babelDateU[1]{\number#1}%
3176 \def\bbl@replace@finish@iii#1{%
3177 \bbl@exp{\def\\#1###1###2###3{\the\toks@}}
3178 \def\bbl@TG@@date{%
3179 \bbl@replace\bbl@toreplace{[ ]}{\babelDateSpace}}%
3180 \bbl@replace\bbl@toreplace{[. ]}{\babelDateDot}}%
3181 \bbl@replace\bbl@toreplace{[d]}{\babelDated{###3}}%
3182 \bbl@replace\bbl@toreplace{[dd]}{\babelDatedd{###3}}%
3183 \bbl@replace\bbl@toreplace{[M]}{\babelDateM{###2}}%
3184 \bbl@replace\bbl@toreplace{[MM]}{\babelDateMM{###2}}%
3185 \bbl@replace\bbl@toreplace{[MMM]}{\babelDateMMM{###2}}%
3186 \bbl@replace\bbl@toreplace{[y]}{\babelDatey{###1}}%
3187 \bbl@replace\bbl@toreplace{[yy]}{\babelDateyy{###1}}%

```



```

3188 \bbl@replace\bbl@toreplace{[yyyy]}\BabelDateyyyy{####1}}%
3189 \bbl@replace\bbl@toreplace{[U]}\BabelDateU{####1}}%
3190 \bbl@replace\bbl@toreplace{[y]}\bbl@datecctr{####1}}%
3191 \bbl@replace\bbl@toreplace{[U]}\bbl@datecctr{####1}}%
3192 \bbl@replace\bbl@toreplace{[m]}\bbl@datecctr{####2}}%
3193 \bbl@replace\bbl@toreplace{[d]}\bbl@datecctr{####3}}%
3194 \bbl@replace@finish@iii\bbl@toreplace}
3195 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3196 \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.

```

3197 \AddToHook{begindocument/before}{%
3198 \let\bbl@normalsf\normalsfcodes
3199 \let\normalsfcodes\relax}
3200 \AtBeginDocument{%
3201 \ifx\bbl@normalsf\@empty
3202 \ifnum\sfcodes\@m
3203 \let\normalsfcodes\frenchspacing
3204 \else
3205 \let\normalsfcodes\nonfrenchspacing
3206 \fi
3207 \else
3208 \let\normalsfcodes\bbl@normalsf
3209 \fi}

```

Transforms.

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

```

3210 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3211 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3212 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3213 #1[#2]{#3}{#4}{#5}}
3214 \begingroup
3215 \catcode\%=12
3216 \catcode\&=14
3217 \gdef\bbl@transforms#1#2#3{%&
3218 \directlua{
3219 local str = [=[#2]=]
3220 str = str:gsub('%d+%.%d+$', '')
3221 token.set_macro('babeltempa', str)
3222 }&
3223 \def\babeltempc{}&
3224 \bbl@xin@{\babeltempa,}{,\bbl@KVP@transforms,}&
3225 \ifin@else
3226 \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&
3227 \fi
3228 \ifin@
3229 \bbl@foreach\bbl@KVP@transforms{%&
3230 \bbl@xin@{: \babeltempa,}{,##1,}&
3231 \ifin@ & font:font:transform syntax
3232 \directlua{
3233 local t = {}
3234 for m in string.gmatch('##1'..' ':'(.)') do
3235 table.insert(t, m)
3236 end
3237 table.remove(t)
3238 token.set_macro('babeltempc', ',font=' .. table.concat(t, ' '))

```

```

3239         }&%
3240     \fi}&%
3241     \in@{.0$}{#2$}&%
3242     \ifin@
3243     \directlua{&% (\attribute) syntax
3244         local str = string.match([[ \bbl@KVP@transforms]],
3245             '%([^(^%([-)%][^%)]-\babeltempa')
3246         if str == nil then
3247             token.set_macro('babeltempb', '')
3248         else
3249             token.set_macro('babeltempb', ',attribute=' .. str)
3250         end
3251     }&%
3252     \toks@{#3}&%
3253     \bbl@exp{&%
3254         \\g@addto@macro\\bbl@release@transforms{&%
3255             \relax &% Closes previous \bbl@transforms@aux
3256             \\bbl@transforms@aux
3257             \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3258             {\language\the\toks@}}&%
3259     \else
3260     \g@addto@macro\bbl@release@transforms{, {#3}}&%
3261     \fi
3262     \fi}
3263 \endgroup

```

4.22. Handle language system

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

```

3264 \def\bbl@provide@lsys#1{%
3265     \bbl@ifunset{bbl@lname@#1}%
3266     {\bbl@load@info{#1}}%
3267     {}%
3268     \bbl@csarg\let{lsys@#1}\empty
3269     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3270     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3271     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3272     \bbl@ifunset{bbl@lname@#1}{}%
3273     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3274     \ifcase\bbl@engine\or\or
3275     \bbl@ifunset{bbl@prehc@#1}{}%
3276     {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3277     {}%
3278     {\ifx\bbl@xenoxyph\undefined
3279         \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3280         \ifx\AtBeginDocument\@notprerr
3281             \expandafter\@secondoftwo % to execute right now
3282         \fi
3283         \AtBeginDocument{%
3284             \bbl@patchfont{\bbl@xenoxyph}%
3285             {\expandafter\select@language\expandafter{\language\the}}%
3286         \fi}}%
3287     \fi
3288     \bbl@csarg\bbl@toglobal{lsys@#1}}

```

4.23. Numerals

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T_EX. Non-digits characters are kept.

```

3289 \def\bbl@setdigits#1#2#3#4#5{%
3290   \bbl@exp{%
3291     \def\<\language name digits>####1{%       i.e., \langdigits
3292       \<bbl@digits@\language name>####1\\\@nil}%
3293       \let\<bbl@cntr@digits@\language name>\<\language name digits>%
3294       \def\<\language name counter>####1{%       i.e., \langcounter
3295         \\\expandafter\<bbl@counter@\language name>%
3296         \\\csname c@####1\endcsname}%
3297         \def\<bbl@counter@\language name>####1{% i.e., \bbl@counter@lang
3298           \\\expandafter\<bbl@digits@\language name>%
3299           \\\number####1\\\@nil}}}%
3300 \def\bbl@tempa##1##2##3##4##5{%
3301   \bbl@exp{%    Wow, quite a lot of hashes! :- (
3302     \def\<bbl@digits@\language name>#####1{%
3303       \\\ifx#####1\\\@nil                % i.e., \bbl@digits@lang
3304       \\\else
3305         \\\ifx0#####1#1%
3306         \\\else\\\ifx1#####1#2%
3307         \\\else\\\ifx2#####1#3%
3308         \\\else\\\ifx3#####1#4%
3309         \\\else\\\ifx4#####1#5%
3310         \\\else\\\ifx5#####1#1%
3311         \\\else\\\ifx6#####1#2%
3312         \\\else\\\ifx7#####1#3%
3313         \\\else\\\ifx8#####1#4%
3314         \\\else\\\ifx9#####1#5%
3315         \\\else#####1%
3316         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3317         \\\expandafter\<bbl@digits@\language name>%
3318         \\\fi}}}%
3319   \bbl@tempa}

```

```

3320 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3321   \ifx\\#1%
3322     \bbl@exp{%
3323       \def\\bbl@tempa####1{%
3324         \<ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3325   \else
3326     \toks@\expandafter{\the\toks@\or #1}%
3327     \expandafter\bbl@buildifcase
3328   \fi}

```

```
3329 \newcommand\localenumberal[2]{\bbl@cs{cntnr#@1@{language name}}{#2}}
3330 \def\bbl@localecntnr#1#2{\localenumberal{#2}{#1}}
3331 \newcommand\localecounter[2]{%
3332   \expandafter\bbl@localecntnr
3333   \expandafter{\number\csname c@#2\endcsname}{#1}}
3334 \def\bbl@alphnumerical#1#2{%
3335   \expandafter\bbl@alphnumerical@i\number#2 76543210\@@{#1}}
3336 \def\bbl@alphnumerical@i#1#2#3#4#5#6#7#8\@@#9{%
3337   \ifcase\@car#8\@nil\or    % Currently <10000, but prepared for bigger
3338     \bbl@alphnumerical@ii{#9}000000#1\or
3339     \bbl@alphnumerical@ii{#9}00000#1#2\or
3340     \bbl@alphnumerical@ii{#9}0000#1#2#3\or
3341     \bbl@alphnumerical@ii{#9}000#1#2#3#4\else
3342     \bbl@alphnum@invalid{>9999}%
```

```

3343 \fi}
3344 \def\bbl@alphanumeric@iii#1#2#3#4#5#6#7#8{%
3345 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@{language}}%
3346 {\bbl@cs{cntr@#1.4@{language}}#5%
3347 \bbl@cs{cntr@#1.3@{language}}#6%
3348 \bbl@cs{cntr@#1.2@{language}}#7%
3349 \bbl@cs{cntr@#1.1@{language}}#8%
3350 \ifnum#6#7#8>\z@
3351 \bbl@ifunset{bbl@cntr@#1.S.321@{language}}{%
3352 {\bbl@cs{cntr@#1.S.321@{language}}}%
3353 \fi}%
3354 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@{language}}}}
3355 \def\bbl@alphnum@invalid#1{%
3356 \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```

3357 \newcommand\BabelUppercaseMapping[3]{%
3358 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3359 \newcommand\BabelTitlecaseMapping[3]{%
3360 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3361 \newcommand\BabelLowercaseMapping[3]{%
3362 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

  The parser for casing and casing.<variant>.
3363 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3364 \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3365 \else
3366 \def\bbl@uftocode#1{\expandafter`\string#1}
3367 \fi
3368 \def\bbl@casemapping#1#2#3{% 1:variant
3369 \def\bbl@tempa##1 ##2{% Loop
3370 \bbl@casemapping@i{##1}%
3371 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3372 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3373 \def\bbl@tempe{0}% Mode (upper/lower...)
3374 \def\bbl@tempc{#3}% Casing list
3375 \expandafter\bbl@tempa\bbl@tempc\@empty}
3376 \def\bbl@casemapping@i#1{%
3377 \def\bbl@tempb{#1}%
3378 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3379 \@nameuse{regex_replace_all:nnN}%
3380 {[\\x{c0}-\\x{ff}][\\x{80}-\\x{bf}]*}{\\0}}\bbl@tempb
3381 \else
3382 \@nameuse{regex_replace_all:nnN}{.}{\\0}}\bbl@tempb
3383 \fi
3384 \expandafter\bbl@casemapping@ii\bbl@tempb\@
3385 \def\bbl@casemapping@ii#1#2#3\@@{%
3386 \in@{#1#3}{<>}% i.e., if <u>, <l>, <t>
3387 \ifin@
3388 \edef\bbl@tempe{%
3389 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3390 \else
3391 \ifcase\bbl@tempe\relax
3392 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3393 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3394 \or
3395 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3396 \or
3397 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3398 \or
3399 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3400 \fi
3401 \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.

```

3402 \def\bbl@localeinfo#1#2{%
3403   \bbl@ifunset{\bbl@info@#2}{#1}%
3404   {\bbl@ifunset{\bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3405    {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3406 \newcommand\localeinfo[1]{%
3407   \ifx*#1\@empty
3408     \bbl@afterelse\bbl@localeinfo{%
3409   \else
3410     \bbl@localeinfo
3411     {\bbl@error{no-ini-info}{}}{}}}%
3412   {#1}%
3413   \fi}
3414 % \@namedef{\bbl@info@name.locale}{lcname}
3415 \@namedef{\bbl@info@tag.ini}{lini}
3416 \@namedef{\bbl@info@name.english}{elname}
3417 \@namedef{\bbl@info@name.opentype}{lname}
3418 \@namedef{\bbl@info@tag.bcp47}{tbc}
3419 \@namedef{\bbl@info@language.tag.bcp47}{lbc}
3420 \@namedef{\bbl@info@tag.opentype}{lotf}
3421 \@namedef{\bbl@info@script.name}{esname}
3422 \@namedef{\bbl@info@script.name.opentype}{sname}
3423 \@namedef{\bbl@info@script.tag.bcp47}{sbcp}
3424 \@namedef{\bbl@info@script.tag.opentype}{sotf}
3425 \@namedef{\bbl@info@region.tag.bcp47}{rbcp}
3426 \@namedef{\bbl@info@variant.tag.bcp47}{vbc}
3427 \@namedef{\bbl@info@extension.t.tag.bcp47}{extt}
3428 \@namedef{\bbl@info@extension.u.tag.bcp47}{extu}
3429 \@namedef{\bbl@info@extension.x.tag.bcp47}{extx}

```

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

```

3430 <<*More package options>> ≡
3431 \DeclareOption{ensureinfo=off}{}
3432 <</More package options>>
3433 \let\BabelEnsureInfo\relax

```

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

```

3434 \newcommand\getlocaleproperty{%
3435   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3436 \def\bbl@getproperty@s#1#2#3{%
3437   \let#1\relax
3438   \def\bbl@elt##1##2##3{%
3439     \bbl@ifsamestring{##1/##2}{#3}%
3440     {\providecommand#1{##3}%
3441      \def\bbl@elt####1####2####3{}}}%
3442   {}}%
3443   \bbl@cs{inidata@#2}}%
3444 \def\bbl@getproperty@x#1#2#3{%
3445   \bbl@getproperty@s{#1}{#2}{#3}%
3446   \ifx#1\relax
3447     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3448   \fi}

```

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

```

3449 \let\bbl@ini@loaded\@empty
3450 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3451 \def\ShowLocaleProperties#1{%
3452   \typeout{}}%
3453   \typeout{*** Properties for language '#1' ***}

```

```

3454 \def\bbl@elt##1##2##3{\typeout{##1/##2 = \unexpanded{##3}}}%
3455 \@nameuse{bbl@inidata@#1}%
3456 \typeout{*****}}

```

4.26. BCP 47 related commands

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

```

3457 \newif\ifbbl@bcppallowed
3458 \bbl@bcppallowedfalse
3459 \def\bbl@autoload@options{@import}
3460 \def\bbl@provide@locale{%
3461   \ifx\babelprovide\@undefined
3462     \bbl@error{base-on-the-fly}{}}}%
3463 \fi
3464 \let\bbl@auxname\language
3465 \ifbbl@bcptoname
3466   \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
3467   {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
3468   \let\locale\language}%
3469 \fi
3470 \ifbbl@bcppallowed
3471   \expandafter\ifx\csname date\language\endcsname\relax
3472     \expandafter
3473     \bbl@bcplookup\language-\@empty-\@empty-\@empty@@
3474     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3475       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
3476       \let\locale\language
3477       \expandafter\ifx\csname date\language\endcsname\relax
3478         \let\bbl@initoload\bbl@bcp
3479         \bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}%
3480         \let\bbl@initoload\relax
3481       \fi
3482       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\locale}%
3483     \fi
3484   \fi
3485 \fi
3486 \expandafter\ifx\csname date\language\endcsname\relax
3487   \IfFileExists{babel-\language.tex}%
3488   {\bbl@exp{\babelprovide[\bbl@autoload@options]{\language}}}%
3489   {}%
3490 \fi}

```

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

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

```

3491 \providecommand\BCPdata{}
3492 \ifx\renewcommand\@undefined\else
3493   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3494   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3495     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}}%
3496     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3497     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3498   \def\bbl@bcpdata@ii#1#2{%
3499     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3500     {\bbl@error{unknown-ini-field}{#1}{}}}%

```

```

3501      {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3502      {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3503 \fi
3504 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3505 \@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.

```

3506 \newcommand\babeladjust[1]{%
3507   \bbl@forkv{#1}{%
3508     \bbl@ifunset{bbl@ADJ@##1@##2}%
3509     {\bbl@cs{ADJ@##1}{##2}}%
3510     {\bbl@cs{ADJ@##1@##2}}}
3511 %
3512 \def\bbl@adjust@lua#1#2{%
3513   \ifvmode
3514     \ifnum\currentgrouplevel=\z@
3515       \directlua{ Babel.#2 }%
3516       \expandafter\expandafter\expandafter@gobble
3517     \fi
3518   \fi
3519   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3520 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3521   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3522 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3523   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3524 \@namedef{bbl@ADJ@bidi.text@on}{%
3525   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3526 \@namedef{bbl@ADJ@bidi.text@off}{%
3527   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3528 \@namedef{bbl@ADJ@bidi.math@on}{%
3529   \let\bbl@noamsmath\empty}
3530 \@namedef{bbl@ADJ@bidi.math@off}{%
3531   \let\bbl@noamsmath\relax}
3532 %
3533 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3534   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3535 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3536   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3537 %
3538 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3539   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3540 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3541   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3542 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3543   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3544 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3545   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3546 \@namedef{bbl@ADJ@justify.arabic@on}{%
3547   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3548 \@namedef{bbl@ADJ@justify.arabic@off}{%
3549   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3550 %
3551 \def\bbl@adjust@layout#1{%
3552   \ifvmode
3553     #1%
3554     \expandafter\expandafter\expandafter@gobble
3555   \fi
3556   {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3557 \@namedef{bbl@ADJ@layout.tabular@on}{%
3558   \ifnum\bbl@tabular@mode=\tw@

```

```

3559 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3560 \else
3561 \chardef\bbl@tabular@mode\@ne
3562 \fi}
3563 \@namedef{bbl@ADJ@layout.tabular@off}{%
3564 \ifnum\bbl@tabular@mode=\tw@
3565 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3566 \else
3567 \chardef\bbl@tabular@mode\z@
3568 \fi}
3569 \@namedef{bbl@ADJ@layout.lists@on}{%
3570 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3571 \@namedef{bbl@ADJ@layout.lists@off}{%
3572 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3573 %
3574 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3575 \bbl@bcpallowedtrue}
3576 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3577 \bbl@bcpallowedfalse}
3578 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3579 \def\bbl@bcp@prefix{#1}}
3580 \def\bbl@bcp@prefix{bcp47-}
3581 \@namedef{bbl@ADJ@autoload.options#1}{%
3582 \def\bbl@autoload@options{#1}}
3583 \def\bbl@autoload@bcptoptions{import}
3584 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3585 \def\bbl@autoload@bcptoptions{#1}}
3586 \newif\ifbbl@bcptname
3587 %
3588 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3589 \bbl@bcptonametrue}
3590 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3591 \bbl@bcptonamefalse}
3592 %
3593 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3594 \directlua{ Babel.ignore_pre_char = function(node)
3595 return (node.lang == \the\csname \@nohyphenation\endcsname)
3596 end }}
3597 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3598 \directlua{ Babel.ignore_pre_char = function(node)
3599 return false
3600 end }}
3601 %
3602 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3603 \def\bbl@ignoreinterchar{%
3604 \ifnum\language=\@nohyphenation
3605 \expandafter\@gobble
3606 \else
3607 \expandafter\@firstofone
3608 \fi}}
3609 \@namedef{bbl@ADJ@interchar.disable@off}{%
3610 \let\bbl@ignoreinterchar\@firstofone}
3611 %
3612 \@namedef{bbl@ADJ@select.write@shift}{%
3613 \let\bbl@restorelastskip\relax
3614 \def\bbl@savelastskip{%
3615 \let\bbl@restorelastskip\relax
3616 \ifvmode
3617 \ifdim\lastskip=\z@
3618 \let\bbl@restorelastskip\nobreak
3619 \else
3620 \bbl@exp{%
3621 \def\\bbl@restorelastskip{%

```



```

3622         \skip@=\the\lastskip
3623         \\nobreak \vskip-\skip@ \vskip\skip@}%
3624     \fi
3625 \fi}}
3626 \@namedef{bbl@ADJ@select.write@keep}{%
3627     \let\bbl@restorelastskip\relax
3628     \let\bbl@savelastskip\relax}
3629 \@namedef{bbl@ADJ@select.write@omit}{%
3630     \AddBabelHook{babel-select}{beforestart}{%
3631         \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3632     \let\bbl@restorelastskip\relax
3633     \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3634 \@namedef{bbl@ADJ@select.encoding@off}{%
3635     \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.

```

3636 << *More package options >> ≡
3637 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3638 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3639 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3640 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3641 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3642 << /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.

```

3643 \bbl@trace{Cross referencing macros}
3644 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3645     \def\@newl@bel#1#2#3{%
3646         {\@safe@activetrue
3647         \bbl@ifunset{#1@#2}%
3648             \relax
3649             {\gdef\@multiplelabels{%
3650                 \latex@warning@no@line{There were multiply-defined labels}}}%
3651                 \latex@warning@no@line{Label `#2' multiply defined}}}%
3652     \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.

```

3653 \CheckCommand*\@testdef[3]{%
3654     \def\reserved@a{#3}%
3655     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3656     \else
3657         \@tempwattrue
3658     \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.

```

3659 \def\@testdef#1#2#3{%
3660   \@safe@activetrue
3661   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3662   \def\bbl@tempb{#3}%
3663   \@safe@activetrue
3664   \ifx\bbl@tempa\relax
3665   \else
3666     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3667   \fi
3668   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3669   \ifx\bbl@tempa\bbl@tempb
3670   \else
3671     \@tempwattrue
3672   \fi}
3673 \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.

```

3674 \bbl@xin@{R}\bbl@opt@safe
3675 \ifin@
3676   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3677   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3678   {\expandafter\strip@prefix\meaning\ref}%
3679 \ifin@
3680   \bbl@redefine\@kernel@ref#1{%
3681     \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3682   \bbl@redefine\@kernel@pageref#1{%
3683     \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3684   \bbl@redefine\@kernel@sref#1{%
3685     \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3686   \bbl@redefine\@kernel@spageref#1{%
3687     \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3688   \else
3689     \bbl@redefinero bust\ref#1{%
3690       \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3691     \bbl@redefinero bust\pageref#1{%
3692       \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3693   \fi
3694 \else
3695   \let\org@ref\ref
3696   \let\org@pageref\pageref
3697 \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.

```

3698 \bbl@xin@{B}\bbl@opt@safe
3699 \ifin@
3700   \bbl@redefine\@citex[#1]#2{%
3701     \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activetrue
3702     \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.)

```
3703 \AtBeginDocument{%
3704   \@ifpackageloaded{natbib}{%
3705     \def\@citex[#1][#2]#3{%
3706       \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activesfalse
3707       \org@citex[#1][#2]{\bbl@tempa}}%
3708     }{}}
```

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

```
3709 \AtBeginDocument{%
3710   \@ifpackageloaded{cite}{%
3711     \def\@citex[#1]#2{%
3712       \@safe@activetrue\org@citex[#1][#2]\@safe@activesfalse}%
3713     }{}}
```

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

```
3714 \bbl@redefine\nocite#1{%
3715   \@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.

```
3716 \bbl@redefine\bibcite{%
3717   \bbl@cite@choice
3718   \bibcite}
```

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

```
3719 \def\bbl@bibcite#1#2{%
3720   \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.

```
3721 \def\bbl@cite@choice{%
3722   \global\let\bibcite\bbl@bibcite
3723   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}}%
3724   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}}%
3725   \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.

```
3726 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3727 \bbl@redefine\@bibitem#1{%
3728   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
3729 \else
3730   \let\org@nocite\nocite
3731   \let\org@citex\citex
```

```

3732 \let\org@bibcite\ibcite
3733 \let\org@@bibitem\@bibitem
3734 \fi

```

5.2. Layout

```

3735 \newcommand\BabelPatchSection[1]{%
3736   \@ifundefined{#1}{}{%
3737     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3738     \@namedef{#1}{%
3739       \ifstar{\bbl@presec@#1}%
3740       {\@dblarg{\bbl@presec@x{#1}}}}%
3741 \def\bbl@presec@x#1[#2]#3{%
3742   \bbl@exp{%
3743     \\select@language@x{\bbl@main@language}%
3744     \\bbl@cs{sspre@#1}%
3745     \\bbl@cs{ss@#1}%
3746     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3747     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
3748     \\select@language@x{\language}}%
3749 \def\bbl@presec@#1#2{%
3750   \bbl@exp{%
3751     \\select@language@x{\bbl@main@language}%
3752     \\bbl@cs{sspre@#1}%
3753     \\bbl@cs{ss@#1}*%
3754     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
3755     \\select@language@x{\language}}%
3756 %
3757 \IfBabelLayout{sectioning}%
3758   {\BabelPatchSection{part}%
3759   \BabelPatchSection{chapter}%
3760   \BabelPatchSection{section}%
3761   \BabelPatchSection{subsection}%
3762   \BabelPatchSection{subsubsection}%
3763   \BabelPatchSection{paragraph}%
3764   \BabelPatchSection{subparagraph}%
3765   \def\babel@toc#1{%
3766     \select@language@x{\bbl@main@language}}}%
3767 \IfBabelLayout{captions}%
3768   {\BabelPatchSection{caption}}}%

```

\BabelFootnote Footnotes.

```

3769 \bbl@trace{Footnotes}
3770 \def\bbl@footnote#1#2#3{%
3771   \@ifnextchar[%
3772     {\bbl@footnote@o{#1}{#2}{#3}}%
3773     {\bbl@footnote@x{#1}{#2}{#3}}%
3774 \long\def\bbl@footnote@x#1#2#3#4{%
3775   \bgroup
3776   \select@language@x{\bbl@main@language}%
3777   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3778   \egroup}
3779 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3780   \bgroup
3781   \select@language@x{\bbl@main@language}%
3782   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3783   \egroup}
3784 \def\bbl@footnotetext#1#2#3{%
3785   \@ifnextchar[%
3786     {\bbl@footnotetext@o{#1}{#2}{#3}}%
3787     {\bbl@footnotetext@x{#1}{#2}{#3}}%
3788 \long\def\bbl@footnotetext@x#1#2#3#4{%
3789   \bgroup

```

```

3790 \select@language@x{\bbl@main@language}%
3791 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3792 \egroup}
3793 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3794 \bgroup
3795 \select@language@x{\bbl@main@language}%
3796 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3797 \egroup}
3798 \def\BabelFootnote#1#2#3#4{%
3799 \ifx\bbl@fn@footnote\@undefined
3800 \let\bbl@fn@footnote\footnote
3801 \fi
3802 \ifx\bbl@fn@footnotetext\@undefined
3803 \let\bbl@fn@footnotetext\footnotetext
3804 \fi
3805 \bbl@ifblank{#2}%
3806 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3807 \namedef{\bbl@stripslash#1text}%
3808 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3809 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3810 \namedef{\bbl@stripslash#1text}%
3811 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3812 \IfBabelLayout{footnotes}%
3813 {\let\bbl@OL@footnote\footnote
3814 \BabelFootnote\footnote\language\{}}%
3815 \BabelFootnote\localfootnote\language\{}}%
3816 \BabelFootnote\mainfootnote\{}}%
3817 {}

```

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.

```

3818 \bbl@trace{Marks}
3819 \IfBabelLayout{sectioning}
3820 {\ifx\bbl@opt@headfoot\@nnil
3821 \g@addto@macro\@resetactivechars{%
3822 \set@typeset@protect
3823 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3824 \let\protect\noexpand
3825 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3826 \edef\thepage{%
3827 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3828 \fi}%
3829 \fi}
3830 {\ifbbl@single\else
3831 \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroobust
3832 \markright#1{%
3833 \bbl@ifblank{#1}%
3834 {\org@markright{}}%
3835 {\toks@{#1}%
3836 \bbl@exp{%
3837 \org@markright{\protect\foreignlanguage{\language}%
3838 {\protect\bbl@restore@actives\the\toks@}}}%

```

\markboth

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

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

```

3839 \ifx\@mkboth\markboth
3840 \def\bbl@tempc{\let\@mkboth\markboth}%
3841 \else
3842 \def\bbl@tempc{%
3843 \fi
3844 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefineroobust
3845 \markboth#1#2{%
3846 \protected@edef\bbl@tempb##1{%
3847 \protect\foreignlanguage
3848 {\language\name}{\protect\bbl@restore@actives##1}}%
3849 \bbl@ifblank{#1}%
3850 {\toks@{}}%
3851 {\toks@\expandafter{\bbl@tempb{#1}}}%
3852 \bbl@ifblank{#2}%
3853 {\@temptokena{}}%
3854 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3855 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}%
3856 \bbl@tempc
3857 \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.

```

3858 \bbl@trace{Preventing clashes with other packages}
3859 \ifx\org@ref\undefined\else
3860 \bbl@xin@{R}\bbl@opt@safe
3861 \ifin@
3862 \AtBeginDocument{%
3863 \@ifpackageloaded{ifthen}{%
3864 \bbl@redefine@long\ifthenelse#1#2#3{%
3865 \let\bbl@temp@pref\pageref
3866 \let\pageref\org@pageref
3867 \let\bbl@temp@ref\ref
3868 \let\ref\org@ref
3869 \@safe@activestrue
3870 \org@ifthenelse{#1}%
3871 {\let\pageref\bbl@temp@pref
3872 \let\ref\bbl@temp@ref
3873 \@safe@activesfalse
3874 #2}%
3875 {\let\pageref\bbl@temp@pref

```

```

3876         \let\ref\bbl@temp@ref
3877         \@safe@activesfalse
3878         #3}%
3879     }%
3880 }{}%
3881 }
3882 \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`.

```

3883 \AtBeginDocument{%
3884   \@ifpackageloaded{varioref}{%
3885     \bbl@redefine\@@vpageref#1[#2]#3{%
3886       \@safe@activestrue
3887       \org@@vpageref{#1}[#2]#3}%
3888     \@safe@activesfalse}%
3889   \bbl@redefine\vrefpagenum#1#2{%
3890     \@safe@activestrue
3891     \org@vrefpagenum{#1}#2}%
3892   \@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.

```

3893   \expandafter\def\csname Ref \endcsname#1{%
3894     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3895   }{}%
3896 }
3897 \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.

```

3898 \AtEndOfPackage{%
3899   \AtBeginDocument{%
3900     \@ifpackageloaded{hhline}%
3901     {\expandafter\ifx\csname normal@char\string\endcsname\relax
3902       \else
3903         \makeatletter
3904         \def\@currname{hhline}\input{hhline.sty}\makeatother
3905       \fi}%
3906     {}}}

```

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

```

3907 \def\substitutefontfamily#1#2#3{%
3908   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3909   \immediate\write15{%
3910     \string\ProvidesFile{#1#2.fd}%
3911     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}}

```

```

3912 \space generated font description file]^^J
3913 \string\DeclareFontFamily{#1}{#2}{ }^^J
3914 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^^J
3915 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^^J
3916 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^^J
3917 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^^J
3918 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^^J
3919 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^^J
3920 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^^J
3921 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^^J
3922 }%
3923 \closeout15
3924 }
3925 \@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`

```

3926 \bbl@trace{Encoding and fonts}
3927 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3928 \newcommand\BabelNonText{TS1,T3,TS3}
3929 \let\org@TeX\TeX
3930 \let\org@LaTeX\LaTeX
3931 \let\ensureascii\@firstofone
3932 \let\asciiencoding\@empty
3933 \AtBeginDocument{%
3934   \def\elt#1{,#1,}%
3935   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3936   \let\elt\relax
3937   \let\bbl@tempb\@empty
3938   \def\bbl@tempc{OT1}%
3939   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3940     \bbl@ifunset{T@#1}{ }\def\bbl@tempb{#1}}}%
3941   \bbl@foreach\bbl@tempa{%
3942     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3943     \ifin@
3944       \def\bbl@tempb{#1}% Store last non-ascii
3945     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3946       \ifin@
3947         \def\bbl@tempc{#1}% Store last ascii
3948       \fi
3949     \fi}%
3950   \ifx\bbl@tempb\@empty\else
3951     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3952     \ifin@
3953       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3954     \fi
3955     \let\asciiencoding\bbl@tempc
3956     \renewcommand\ensureascii[1]{%
3957       {\fontencoding\asciiencoding}\selectfont#1}}%
3958     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3959     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3960   \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.

```
3961 \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.

```
3962 \AtBeginDocument{%
3963   \@ifpackageloaded{fontspec}%
3964   {\xdef\latinencoding{%
3965     \ifx\UTFencname\@undefined
3966       EU\ifcase\bbl@engine\or2\or1\fi
3967     \else
3968       \UTFencname
3969     \fi}}%
3970   {\gdef\latinencoding{OT1}%
3971     \ifx\cf@encoding\bbl@t@one
3972       \xdef\latinencoding{\bbl@t@one}%
3973     \else
3974       \def\@elt#1{,#1,}%
3975       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3976       \let\@elt\relax
3977       \bbl@xin@{,T1,}\bbl@tempa
3978       \ifin@
3979         \xdef\latinencoding{\bbl@t@one}%
3980       \fi
3981     \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.

```
3982 \DeclareRobustCommand{\latintext}{%
3983   \fontencoding{\latinencoding}\selectfont
3984   \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.

```
3985 \ifx\@undefined\DeclareTextFontCommand
3986   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3987 \else
3988   \DeclareTextFontCommand{\textlatin}{\latintext}
3989 \fi
```

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

```
3990 \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.

```

3991 \bbl@trace{Loading basic (internal) bidi support}
3992 \ifodd\bbl@engine
3993 \else % Any xe+lua bidi
3994   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3995     \bbl@error{bidi-only-lua}{}}{}%
3996     \let\bbl@beforeforeign\leavevmode
3997     \AtEndOfPackage{%
3998       \EnableBabelHook{babel-bidi}%
3999       \bbl@xebidipar}
4000 \fi\fi
4001 \def\bbl@loadxebidi#1{%
4002   \ifx\RTLfootnotetext\@undefined
4003     \AtEndOfPackage{%
4004       \EnableBabelHook{babel-bidi}%
4005       \ifx\fontspec\@undefined
4006         \usepackage{fontspec}% bidi needs fontspec
4007       \fi
4008       \usepackage#1{bidi}%
4009       \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4010       \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4011         \ifnum\@nameuse\bbl@wdir\@languagename=\tw@ % 'AL' bidi
4012           \bbl@digitsdotdash % So ignore in 'R' bidi
4013         \fi}}%
4014   \fi}
4015 \ifnum\bbl@bidimode>200 % Any xe bidi=
4016   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4017     \bbl@tentative{bidi=bidi}
4018     \bbl@loadxebidi{}
4019   \or
4020     \bbl@loadxebidi{[rldocument]}
4021   \or
4022     \bbl@loadxebidi{}
4023   \fi
4024 \fi
4025 \fi
4026 \ifnum\bbl@bidimode=\@ne % bidi=default
4027   \let\bbl@beforeforeign\leavevmode
4028   \ifodd\bbl@engine % lua
4029     \newattribute\bbl@attr@dir
4030     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4031     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4032   \fi
4033   \AtEndOfPackage{%
4034     \EnableBabelHook{babel-bidi}% pdf/lua/xe
4035     \ifodd\bbl@engine\else % pdf/xe
4036       \bbl@xebidipar
4037     \fi}
4038 \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.

```

4039 \bbl@trace{Macros to switch the text direction}

```

```

4040 \def\bbl@alscripts{%
4041   ,Arabic,Syriac,Thaana,Hanifi Rohingya,Hanifi,Sogdian,}
4042 \def\bbl@rscripts{%
4043   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4044   Hatran,Hebrew,Imperial Aramaic,Inscriptional Pahlavi,%
4045   Inscriptional Parthian,Kharoshthi,Lydian,Mandaic,Manichaean,%
4046   Mende Kikakui,Meroitic Cursive,Meroitic Hieroglyphs,Nabataean,%
4047   Nko,Old Hungarian,Old North Arabian,Old Sogdian,%
4048   Old South Arabian,Old Turkic,Old Uyghur,Palmyrene,Phoenician,%
4049   Psalter Pahlavi,Samaritan,Yezidi,Mandaean,%
4050   Meroitic,N'Ko,Orkhon,Todhri}
4051 %
4052 \def\bbl@provide@dirs#1{%
4053   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4054   \ifin@
4055     \global\bbl@csarg\chardef{wdir@#1}\@ne
4056     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4057     \ifin@
4058       \global\bbl@csarg\chardef{wdir@#1}\tw@
4059       \fi
4060   \else
4061     \global\bbl@csarg\chardef{wdir@#1}\z@
4062   \fi
4063   \ifodd\bbl@engine
4064     \bbl@csarg\ifcase{wdir@#1}%
4065       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4066     \or
4067       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4068     \or
4069       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4070     \fi
4071   \fi}
4072 %
4073 \def\bbl@switchdir{%
4074   \bbl@ifunset{\bbl@lsys{\languagename}}{\bbl@provide@lsys{\languagename}}{}}%
4075   \bbl@ifunset{\bbl@wdir{\languagename}}{\bbl@provide@dirs{\languagename}}{}}%
4076   \bbl@expf{\bbl@setdirs\bbl@cl{wdir}}}%
4077 \def\bbl@setdirs#1{%
4078   \ifcase\bbl@select@type
4079     \bbl@bodydir{#1}%
4080     \bbl@pardir{#1}% <- Must precede \bbl@texdir
4081   \fi
4082   \bbl@texdir{#1}}
4083 \ifnum\bbl@bidimode>\z@
4084   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4085   \DisableBabelHook{babel-bidi}
4086 \fi

```

Now the engine-dependent macros.

```

4087 \ifodd\bbl@engine % luatex=1
4088 \else % pdftex=0, xetex=2
4089   \newcount\bbl@dirlevel
4090   \chardef\bbl@thetexdir\z@
4091   \chardef\bbl@thepardir\z@
4092   \def\bbl@texdir#1{%
4093     \ifcase#1\relax
4094       \chardef\bbl@thetexdir\z@
4095       \@nameuse{setlatin}%
4096       \bbl@texdir@i\beginL\endL
4097     \else
4098       \chardef\bbl@thetexdir\@ne
4099       \@nameuse{setnonlatin}%
4100       \bbl@texdir@i\beginR\endR

```

```

4101 \fi}
4102 \def\bbl@textdir@i#1#2{%
4103 \ifhmode
4104 \ifnum\currentgrouplevel>\z@
4105 \ifnum\currentgrouplevel=\bbl@dirlevel
4106 \bbl@error{multiple-bidi}{}}{}%
4107 \bgroup\aftergroup#2\aftergroup\egroup
4108 \else
4109 \ifcase\currentgrouptype\or % 0 bottom
4110 \aftergroup#2% 1 simple {}
4111 \or
4112 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4113 \or
4114 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4115 \or\or\or % vbox vtop align
4116 \or
4117 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4118 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4119 \or
4120 \aftergroup#2% 14 \begingroup
4121 \else
4122 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4123 \fi
4124 \fi
4125 \bbl@dirlevel\currentgrouplevel
4126 \fi
4127 #1%
4128 \fi}
4129 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4130 \let\bbl@bodydir\@gobble
4131 \let\bbl@pagedir\@gobble
4132 \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).

```

4133 \def\bbl@xebidipar{%
4134 \let\bbl@xebidipar\relax
4135 \TeXeTstate\@ne
4136 \def\bbl@xeeverypar{%
4137 \ifcase\bbl@thepardir
4138 \ifcase\bbl@thetextdir\else\beginR\fi
4139 \else
4140 {\setbox\z@\lastbox\beginR\box\z@}%
4141 \fi}%
4142 \AddToHook{para/begin}{\bbl@xeeverypar}}
4143 \ifnum\bbl@bidimode>200 % Any xe bidi=
4144 \let\bbl@textdir@i\@gobbletwo
4145 \let\bbl@xebidipar\@empty
4146 \AddBabelHook{bidi}{foreign}{%
4147 \ifcase\bbl@thetextdir
4148 \BabelWrapText{\LR{##1}}%
4149 \else
4150 \BabelWrapText{\RL{##1}}%
4151 \fi}
4152 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4153 \fi
4154 \fi

```

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

```

4155 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}}
4156 \AtBeginDocument{%
4157 \ifx\pdfstringdefDisableCommands\undefined\else
4158 \ifx\pdfstringdefDisableCommands\relax\else

```

```

4159 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4160 \fi
4161 \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.

```

4162 \bbl@trace{Local Language Configuration}
4163 \ifx\loadlocalcfg\undefined
4164 \ifpackagewith{babel}{noconfigs}%
4165 {\let\loadlocalcfg@gobble}%
4166 {\def\loadlocalcfg#1{%
4167 \InputIfFileExists{#1.cfg}%
4168 {\typeout{*****^J%
4169 * Local config file #1.cfg used^^J%
4170 *}}}%
4171 \@empty}}
4172 \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).

```

4173 \bbl@trace{Language options}
4174 \def\BabelDefinitionFile#1#2#3{}
4175 \let\bbl@afterlang\relax
4176 \let\BabelModifiers\relax
4177 \let\bbl@loaded\@empty
4178 \def\bbl@load@language#1{%
4179 \InputIfFileExists{#1.ldf}%
4180 {\edef\bbl@loaded{\CurrentOption
4181 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4182 \expandafter\let\expandafter\bbl@afterlang
4183 \csname\CurrentOption.ldf-h@k\endcsname
4184 \expandafter\let\expandafter\BabelModifiers
4185 \csname bbl@mod@\CurrentOption\endcsname
4186 \bbl@exp{\AtBeginDocument{%
4187 \bbl@usehooks@lang{\CurrentOption}{\begin{document}}{\CurrentOption}}}%
4188 {\bbl@error{unknown-package-option}}}}

```

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

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

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

```

4189 \ifx\GetDocumentProperties\undefined\else
4190 \let\bbl@beforeforeign\leavevmode
4191 \edef\bbl@metalang{\GetDocumentProperties{document/lang}}%
4192 \ifx\bbl@metalang\@empty\else
4193 \begingroup
4194 \expandafter

```

```

4195 \bbl@bcplookup\bbl@metalang-\@empty-\@empty-\@empty@@
4196 \ifx\bbl@bcp\relax
4197 \ifx\bbl@opt@main\@nnil
4198 \bbl@error{no-locale-for-meta}{\bbl@metalang}{\{}}%
4199 \fi
4200 \else
4201 \bbl@read@ini{\bbl@bcp}\m@ne
4202 \xdef\bbl@language@opts{\bbl@language@opts,\language\name}%
4203 \ifx\bbl@opt@main\@nnil
4204 \global\let\bbl@opt@main\language\name
4205 \fi
4206 \bbl@info{Passing \language\name space to babel}%
4207 \fi
4208 \endgroup
4209 \fi
4210 \fi
4211 \ifx\bbl@opt@config\@nnil
4212 \@ifpackagewith{babel}{noconfigs}{\%}
4213 {\InputIfFileExists{bblopts.cfg}%
4214 {\bbl@warning{Configuration files are deprecated, as\\%
4215 they can break document portability.\\%
4216 Reported}%
4217 \typeout{*****^J%
4218 * Local config file bblopts.cfg used^^J%
4219 *}}%
4220 {\}%}
4221 \else
4222 \InputIfFileExists{\bbl@opt@config.cfg}%
4223 {\bbl@warning{Configuration files are deprecated, as\\%
4224 they can break document portability.\\%
4225 Reported}%
4226 \typeout{*****^J%
4227 * Local config file \bbl@opt@config.cfg used^^J%
4228 *}}%
4229 {\bbl@error{config-not-found}{\{}}{\}%}
4230 \fi

```

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

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

```

4231 %%%
4232 \def\BabelBeforeIni#1#2{%
4233 \def\bbl@tempa{\m}% <- Default if no \BDefFile
4234 \let\bbl@tempb\@empty
4235 #2%
4236 \edef\bbl@toload{%
4237 \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4238 \bbl@toload@last}%
4239 \edef\bbl@toload@last{0/\bbl@tempa//\CurrentOption//\#1/\bbl@tempb}}
4240 \def\BabelDefinitionFile#1#2#3{%
4241 \def\bbl@tempa{#1}\def\bbl@tempb{#2}%
4242 \@namedef{\bbl@preldf\CurrentOption}{#3}%
4243 \endinput}%

```

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

```
4244 \def\bbl@tempf{,}
4245 \bbl@foreach\@raw@classoptionslist{%
4246   \in@{=}{#1}%
4247   \ifin@else
4248     \edef\bbl@tempf{\bbl@tempf\zap@space#1 \@empty,}%
4249   \fi}
```

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

```
4250 \let\bbl@toload\@empty
4251 \let\bbl@toload@last\@empty
4252 \let\bbl@unkopt\@gobble %% <- Ugly
4253 \edef\bbl@tempc{%
4254   \bbl@tempf,@,\bbl@language@opts
4255   \ifx\bbl@opt@main\@nnil\else,\bbl@opt@main\fi}
4256 %
4257 \bbl@foreach\bbl@tempc{%
4258   \in@{@@}{#1}% <- Ugly
4259   \ifin@
4260     \def\bbl@unkopt##1{%
4261       \DeclareOption{##1}{\bbl@error{unknown-package-option}}{}}}%
4262   \else
4263     \def\CurrentOption{#1}%
4264     \bbl@xin@{//#1//}{\bbl@toload@last}% Collapse consecutive
4265     \ifin@else
4266       \lowercase{\InputIfFileExists{babel-#1.tex}}{ }{%
4267         \IfFileExists{#1.ldf}%
4268           {\edef\bbl@toload{%
4269             \ifx\bbl@toload\@empty\else\bbl@toload,\fi
4270             \bbl@toload@last}%
4271           \edef\bbl@toload@last{0/0//\CurrentOption//und/#1}}%
4272         {\bbl@unkopt{#1}}}%
4273     \fi
4274   \fi}
```

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

```
4275 \ifx\bbl@opt@main\@nnil
4276   \ifx\bbl@toload@last\@empty
4277     \def\bbl@toload@last{0/0//nil//und-x-nil-nil}
4278     \bbl@info{%
4279       You haven't specified a language as a class or package\%
4280       option. I'll load 'nil'. Reported}
4281   \fi
4282 \else
4283   \let\bbl@tempc\@empty
4284   \bbl@foreach\bbl@toload{%
4285     \bbl@xin@{//#1//}{\bbl@opt@main//}{#1}%
4286     \ifin@else
4287       \bbl@add@list\bbl@tempc{#1}%
4288     \fi}
4289   \let\bbl@toload\bbl@tempc
4290 \fi
4291 \edef\bbl@toload{\bbl@toload,1\bbl@toload@last}
```

Finally, load the 'ini' file or the pair 'ini'/'ldf' file. Babel resorts to its own mechanism, not the default one based on `\ProcessOptions` (which is still present to make some internal clean-up). First,

handle provide=! and friends (with a recursive call if they are present), and then provide=* and friend. \count@ is used as flag: 0 if 'ini', 1 if 'ldf'.

```

4292 \let\bbl@tempb\@empty
4293 \def\bbl@tempc#1/#2//#3//#4/#5\@{
4294   % \message{^^J*****#1/#2// #3 // #4/#5}%
4295   \count@\z@
4296   \ifnum#2=\@m % if no \BabelDefinitionFile
4297     \ifnum#1=\z@ % not main. -- % if provide+=!, provide*=!
4298       \ifodd\bbl@ldfflag>\@ne\bbl@tempc 0/0//#3//#4/#3\@
4299       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4300       \fi
4301     \else % 10 = main -- % if provide=!, provide*=!
4302       \ifodd\bbl@ldfflag\bbl@tempc 10/0//#3//#4/#3\@
4303       \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4304       \fi
4305     \fi
4306   \else
4307     \ifnum#1=\z@ % not main
4308       \ifnum\bbl@iniflag>\@ne\else % if 0, provide
4309         \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4310         \fi
4311       \else % 10 = main
4312         \ifodd\bbl@iniflag\else % if provide+, provide*
4313         \ifcase#2\count@\@ne\else\ifcase\bbl@engine\count@\@ne\fi\fi
4314         \fi
4315       \fi
4316       \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4317     \fi}

```

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

```

4318 \def\bbl@tempd#1#2#3#4#5{
4319   \DeclareOption{#3}{
4320     \ifcase\count@
4321       \bbl@exp{\bbl@add\bbl@tempb{
4322         \@nameuse{\bbl@preini#3}%
4323         \bbl@ldfinit % todo: prevent a second load
4324         \def\CurrentOption{#3}%
4325         \bbl@babelprovide[import=#4,\ifnum#1=\z@\else\bbl@opt@provide,main\fi]{#3}%
4326         \bbl@afterldf}}%
4327     \else
4328       \bbl@add\bbl@tempb{
4329         \def\CurrentOption{#3}%
4330         \let\localename\CurrentOption
4331         \let\language\localename
4332         \def\BabelIniTag{#4}%
4333         \@nameuse{\bbl@preldf#3}%
4334         \begingroup
4335           \bbl@id@assign
4336           \bbl@read@ini{\BabelIniTag}0%
4337         \endgroup
4338         \bbl@load@language{#5}}%
4339       \fi}
4340   \NewHook{babel/presets}
4341   \UseHook{babel/presets}
4342   \bbl@foreach\bbl@to load{\bbl@tempc#1\@}
4343   %
4344   \def\AfterBabelLanguage#1{
4345     \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4346   \bbl@tempb
4347   \DeclareOption*{}
4348   \ProcessOptions
4349   %%%%%%%%%%%

```



```

4350 \bbl@exp{%
4351   \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{\begindocument}{}}}%
4352 \def\AfterBabelLanguage{\bbl@error{late-after-babel}}{}{}
4353 \end{package}

```

6. The kernel of Babel

The kernel of the babel system is currently stored in `babel.def`. The file `babel.def` contains most of the code. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns.

Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX , some of it is for the \LaTeX case only.

Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don't load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

A proxy file for `switch.def`

```

4354 \*kernel
4355 \let\bbl@onlyswitch\@empty
4356 \input babel.def
4357 \let\bbl@onlyswitch\@undefined
4358 \end{kernel}

```

7. Error messages

They are loaded when `\bbl@error` is first called. To save space, the main code just identifies them with a tag, and messages are stored in a separate file. Since it can be loaded anywhere, you make sure some catcodes have the right value, although those for `\`, ```, `^`, `M`, `%` and `=` are reset before loading the file.

```

4359 \*errors
4360 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4361 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4362 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4363 \catcode`\@=11 \catcode`\^=7
4364 %
4365 \ifx\MessageBreak\@undefined
4366   \gdef\bbl@error@i#1#2{%
4367     \begingroup
4368       \newlinechar=^^J
4369       \def\{^J(babel) }%
4370       \errhelp{#2}\errmessage{\{#1}%
4371     \endgroup}
4372 \else
4373   \gdef\bbl@error@i#1#2{%
4374     \begingroup
4375       \def\{\MessageBreak}%
4376       \PackageError{babel}{#1}{#2}%
4377     \endgroup}
4378 \fi
4379 \def\bbl@errmessage#1#2#3{%
4380   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4381     \bbl@error@i{#2}{#3}}
4382 % Implicit #2#3#4:
4383 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4384 %
4385 \bbl@errmessage{not-yet-available}
4386   {Not yet available}%
4387   {Find an armchair, sit down and wait}
4388 \bbl@errmessage{bad-package-option}%
4389   {Bad option '#1=#2'. Either you have misspelled the\%

```

```

4390     key or there is a previous setting of '#1'. Valid\\%
4391     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4392     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4393     {See the manual for further details.}
4394 \bbl@errmessage{base-on-the-fly}
4395     {For a language to be defined on the fly 'base'\\%
4396     is not enough, and the whole package must be\\%
4397     loaded. Either delete the 'base' option or\\%
4398     request the languages explicitly}%
4399     {See the manual for further details.}
4400 \bbl@errmessage{undefined-language}
4401     {You haven't defined the language '#1' yet.\\%
4402     Perhaps you misspelled it or your installation\\%
4403     is not complete}%
4404     {Your command will be ignored, type <return> to proceed}
4405 \bbl@errmessage{invalid-ini-name}
4406     {'#1' not valid with the 'ini' mechanism.\MessageBreak
4407     I think you want '#2' instead.\MessageBreak
4408     I'll stop immediately}%
4409     {See the babel manual for the available\MessageBreak
4410     locales with 'provide'}
4411 \bbl@errmessage{shorthand-is-off}
4412     {I can't declare a shorthand turned off (\string#2)}
4413     {Sorry, but you can't use shorthands which have been\\%
4414     turned off in the package options}
4415 \bbl@errmessage{not-a-shorthand}
4416     {The character '\string #1' should be made a shorthand character;\\%
4417     add the command \string\usesshorthands\string{#1\string} to
4418     the preamble.\\%
4419     I will ignore your instruction}%
4420     {You may proceed, but expect unexpected results}
4421 \bbl@errmessage{not-a-shorthand-b}
4422     {I can't switch '\string#2' on or off--not a shorthand\\%
4423     This character is not a shorthand. Maybe you made\\%
4424     a typing mistake?}%
4425     {I will ignore your instruction.}
4426 \bbl@errmessage{unknown-attribute}
4427     {The attribute #2 is unknown for language #1.}%
4428     {Your command will be ignored, type <return> to proceed}
4429 \bbl@errmessage{missing-group}
4430     {Missing group for string \string#1}%
4431     {You must assign strings to some category, typically\\%
4432     captions or extras, but you set none}
4433 \bbl@errmessage{only-lua-xe}
4434     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4435     {Consider switching to these engines.}
4436 \bbl@errmessage{only-lua}
4437     {This macro is available only in LuaLaTeX}%
4438     {Consider switching to that engine.}
4439 \bbl@errmessage{unknown-provide-key}
4440     {Unknown key '#1' in \string\babelprovide}%
4441     {See the manual for valid keys}%
4442 \bbl@errmessage{unknown-mapfont}
4443     {Option '\bbl@KVP@mapfont' unknown for\\%
4444     mapfont. Use 'direction'}%
4445     {See the manual for details.}
4446 \bbl@errmessage{no-ini-file}
4447     {There is no ini file for the requested language\\%
4448     (#1: \language). Perhaps you misspelled it or your\\%
4449     installation is not complete}%
4450     {Fix the name or reinstall babel.}
4451 \bbl@errmessage{digits-is-reserved}
4452     {The counter name 'digits' is reserved for mapping\\%

```

```

4453     decimal digits}%
4454     {Use another name.}
4455 \bbl@errmessage{limit-two-digits}
4456     {Currently two-digit years are restricted to the\\
4457     range 0-9999}%
4458     {There is little you can do. Sorry.}
4459 \bbl@errmessage{alphabetic-too-large}
4460 {Alphabetic numeral too large (#1)}%
4461 {Currently this is the limit.}
4462 \bbl@errmessage{no-ini-info}
4463     {I've found no info for the current locale.\\%
4464     The corresponding ini file has not been loaded\\%
4465     Perhaps it doesn't exist}%
4466     {See the manual for details.}
4467 \bbl@errmessage{unknown-ini-field}
4468     {Unknown field '#1' in \string\BCPdata.\\%
4469     Perhaps you misspelled it}%
4470     {See the manual for details.}
4471 \bbl@errmessage{unknown-locale-key}
4472     {Unknown key for locale '#2':\\%
4473     #3\\%
4474     \string#1 will be set to \string\relax}%
4475     {Perhaps you misspelled it.}%
4476 \bbl@errmessage{adjust-only-vertical}
4477     {Currently, #1 related features can be adjusted only\\%
4478     in the main vertical list}%
4479     {Maybe things change in the future, but this is what it is.}
4480 \bbl@errmessage{layout-only-vertical}
4481     {Currently, layout related features can be adjusted only\\%
4482     in vertical mode}%
4483     {Maybe things change in the future, but this is what it is.}
4484 \bbl@errmessage{bidi-only-lua}
4485     {The bidi method 'basic' is available only in\\%
4486     luatex. I'll continue with 'bidi=default', so\\%
4487     expect wrong results. With xetex, try bidi=bidi}%
4488     {See the manual for further details.}
4489 \bbl@errmessage{multiple-bidi}
4490     {Multiple bidi settings inside a group}%
4491     {I'll insert a new group, but expect wrong results.}
4492 \bbl@errmessage{unknown-package-option}
4493     {Unknown option '\CurrentOption'.\\%
4494     Suggested actions:\\%
4495     * Make sure you haven't misspelled it\\%
4496     * Check in the babel manual that it's supported\\%
4497     * If supported and it's a language, you may\\%
4498     \space\space need in some distributions a separate\\%
4499     \space\space installation\\%
4500     * If installed, check there isn't an old\\%
4501     \space\space version of the required files in your system}
4502     {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4503     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4504     headfoot=, strings=, config=, hyphenmap=, or a language name.}
4505 \bbl@errmessage{config-not-found}
4506     {Local config file '\bbl@opt@config.cfg' not found.\\%
4507     Suggested actions:\\%
4508     * Make sure you haven't misspelled it in config=\\%
4509     * Check it exists and it's in the correct path}%
4510     {Perhaps you misspelled it.}
4511 \bbl@errmessage{late-after-babel}
4512     {Too late for \string\AfterBabelLanguage}%
4513     {Languages have been loaded, so I can do nothing}
4514 \bbl@errmessage{double-hyphens-class}
4515     {Double hyphens aren't allowed in \string\babelcharclass\\%

```

```

4516     because it's potentially ambiguous}%
4517     {See the manual for further info}
4518 \bbl@errmessage{unknown-interchar}
4519     {'#1' for '\language' cannot be enabled.\\%
4520     Maybe there is a typo}%
4521     {See the manual for further details.}
4522 \bbl@errmessage{unknown-interchar-b}
4523     {'#1' for '\language' cannot be disabled.\\%
4524     Maybe there is a typo}%
4525     {See the manual for further details.}
4526 \bbl@errmessage{charproperty-only-vertical}
4527     {\string\babelcharproperty\space can be used only in\\%
4528     vertical mode (preamble or between paragraphs)}%
4529     {See the manual for further info}
4530 \bbl@errmessage{unknown-char-property}
4531     {No property named '#2'. Allowed values are\\%
4532     direction (bc), mirror (bmg), and linebreak (lb)}%
4533     {See the manual for further info}
4534 \bbl@errmessage{bad-transform-option}
4535     {Bad option '#1' in a transform.\\%
4536     I'll ignore it but expect more errors}%
4537     {See the manual for further info.}
4538 \bbl@errmessage{font-conflict-transforms}
4539     {Transforms cannot be re-assigned to different\\%
4540     fonts. The conflict is in '\bbl@kv@label'.\\%
4541     Apply the same fonts or use a different label}%
4542     {See the manual for further details.}
4543 \bbl@errmessage{transform-not-available}
4544     {'#1' for '\language' cannot be enabled.\\%
4545     Maybe there is a typo or it's a font-dependent transform}%
4546     {See the manual for further details.}
4547 \bbl@errmessage{transform-not-available-b}
4548     {'#1' for '\language' cannot be disabled.\\%
4549     Maybe there is a typo or it's a font-dependent transform}%
4550     {See the manual for further details.}
4551 \bbl@errmessage{year-out-range}
4552     {Year out of range.\\%
4553     The allowed range is #1}%
4554     {See the manual for further details.}
4555 \bbl@errmessage{only-pdfTeX-lang}
4556     {The '#1' ldf style doesn't work with #2,\\%
4557     but you can use the ini locale instead.\\%
4558     Try adding 'provide=*' to the option list. You may\\%
4559     also want to set 'bidi=' to some value}%
4560     {See the manual for further details.}
4561 \bbl@errmessage{hyphenmins-args}
4562     {\string\babelhyphenmins\ accepts either the optional\\%
4563     argument or the star, but not both at the same time}%
4564     {See the manual for further details.}
4565 \bbl@errmessage{no-locale-for-meta}
4566     {There isn't currently a locale for the 'lang' requested\\%
4567     in the PDF metadata ('#1'). To fix it, you can\\%
4568     set explicitly a similar language (using the same\\%
4569     script) with the key main= when loading babel. If you\\%
4570     continue, I'll fallback to the 'nil' language, with\\%
4571     tag 'und' and script 'Latn', but expect a bad font\\%
4572     rendering with other scripts. You may also need set\\%
4573     explicitly captions and date, too}%
4574     {See the manual for further details.}
4575 </errors>
4576 <*patterns>

```

8. Loading hyphenation patterns

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

```
4577 <@Make sure ProvidesFile is defined>
4578 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4579 \xdef\bbl@format{\jobname}
4580 \def\bbl@version{<@version@>}
4581 \def\bbl@date{<@date@>}
4582 \ifx\AtBeginDocument\undefined
4583   \def\@empty{}
4584 \fi
4585 <@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.

```
4586 \def\process@line#1#2 #3 #4 {%
4587   \ifx=#1%
4588     \process@synonym{#2}%
4589   \else
4590     \process@language{#1#2}{#3}{#4}%
4591   \fi
4592   \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.

```
4593 \toks@{}
4594 \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.

```
4595 \def\process@synonym#1{%
4596   \ifnum\last@language=\m@ne
4597     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4598   \else
4599     \expandafter\chardef\csname l@#1\endcsname\last@language
4600     \wlog{\string\l@#1=\string\language\the\last@language}%
4601     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4602       \csname\language\hyphenmins\endcsname
4603     \let\bbl@elt\relax
4604     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4605   \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`. $\text{\texttt{T\TeX}}$ does not keep track of these assignments. Therefore we try to detect such assignments and store them in the `\<language>hyphenmins` macro. When no assignments were made we provide a default setting.

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

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

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

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

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

```

4606 \def\process@language#1#2#3{%
4607   \expandafter\addlanguage\csname l@#1\endcsname
4608   \expandafter\language\csname l@#1\endcsname
4609   \edef\language#1{%
4610     \bbl@hook@everylanguage{#1}%
4611     % > luatex
4612     \bbl@get@enc#1::\@@@
4613   \begingroup
4614     \lefthyphenmin\m@ne
4615     \bbl@hook@loadpatterns{#2}%
4616     % > luatex
4617     \ifnum\lefthyphenmin=\m@ne
4618     \else
4619       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4620         \the\lefthyphenmin\the\righthyphenmin}%
4621     \fi
4622   \endgroup
4623   \def\bbl@tempa{#3}%
4624   \ifx\bbl@tempa\@empty\else
4625     \bbl@hook@loadexceptions{#3}%
4626     % > luatex
4627   \fi
4628   \let\bbl@elt\relax
4629   \edef\bbl@languages{%
4630     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4631   \ifnum\the\language=\z@
4632     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4633       \set@hyphenmins\tw@\thr@@\relax
4634     \else
4635       \expandafter\expandafter\expandafter\set@hyphenmins
4636       \csname #1hyphenmins\endcsname
4637     \fi
4638     \the\toks@
4639     \toks@{}%
4640   \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.

```

4641 \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.

```

4642 \def\bbl@hook@everylanguage#1{}
4643 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4644 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4645 \def\bbl@hook@loadkernel#1{%
4646   \def\addlanguage{\csname newlanguage\endcsname}%

```

```

4647 \def\adddialect##1##2{%
4648   \global\chardef##1##2\relax
4649   \log{\string##1 = a dialect from \string\language##2}}%
4650 \def\iflanguage##1{%
4651   \expandafter\ifx\csname l@##1\endcsname\relax
4652     \@nolanner{##1}%
4653   \else
4654     \ifnum\csname l@##1\endcsname=\language
4655       \expandafter\expandafter\expandafter\@firstoftwo
4656     \else
4657       \expandafter\expandafter\expandafter\@secondoftwo
4658     \fi
4659   \fi}%
4660 \def\providehyphenmins##1##2{%
4661   \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4662     \@namedef{##1hyphenmins}{##2}%
4663   \fi}%
4664 \def\set@hyphenmins##1##2{%
4665   \lefthyphenmin##1\relax
4666   \righthyphenmin##2\relax}%
4667 \def\selectlanguage{%
4668   \errhelp{Selecting a language requires a package supporting it}%
4669   \errmessage{No multilingual package has been loaded}}%
4670 \let\foreignlanguage\selectlanguage
4671 \let\otherlanguage\selectlanguage
4672 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4673 \def\bbl@usehooks##1##2{%
4674   \def\setlocale{%
4675     \errhelp{Find an armchair, sit down and wait}%
4676     \errmessage{(babel) Not yet available}}%
4677   \let\uselocale\setlocale
4678   \let\locale\setlocale
4679   \let\selectlocale\setlocale
4680   \let\localename\setlocale
4681   \let\textlocale\setlocale
4682   \let\textlanguage\setlocale
4683   \let\languagetext\setlocale}
4684 \begingroup
4685   \def\AddBabelHook#1#2{%
4686     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4687       \def\next{\toks1}%
4688     \else
4689       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4690     \fi
4691     \next}
4692   \ifx\directlua\@undefined
4693     \ifx\XeTeXinputencoding\@undefined\else
4694       \input xebabel.def
4695     \fi
4696   \else
4697     \input luababel.def
4698   \fi
4699   \openin1 = babel-\bbl@format.cfg
4700   \ifeof1
4701   \else
4702     \input babel-\bbl@format.cfg\relax
4703   \fi
4704   \closein1
4705 \endgroup
4706 \bbl@hook@loadkernel{switch.def}

```

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

```
4707 \openin1 = language.dat
```

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

```
4708 \def\language{english}%
4709 \ifeof1
4710   \message{I couldn't find the file language.dat,\space
4711           I will try the file hyphen.tex}
4712   \input hyphen.tex\relax
4713   \chardef\l@english\z@
4714 \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`.

```
4715 \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.

```
4716 \loop
4717   \endlinechar@m@ne
4718   \read1 to \bbl@line
4719   \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.

```
4720   \if T\ifeof1F\fi T\relax
4721   \ifx\bbl@line\empty\else
4722     \edef\bbl@line{\bbl@line\space\space\space}%
4723     \expandafter\process@line\bbl@line\relax
4724   \fi
4725 \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.

```
4726 \begingroup
4727   \def\bbl@elt#1#2#3#4{%
4728     \global\language=#2\relax
4729     \gdef\language{#1}%
4730     \def\bbl@elt##1##2##3##4{}}%
4731   \bbl@languages
4732 \endgroup
4733 \fi
4734 \closein1
```

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

```
4735 \if\the\toks@\else
4736   \errhelp{language.dat loads no language, only synonyms}
4737   \errmessage{Orphan language synonym}
4738 \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.

```
4739 \let\bbl@line\@undefined
4740 \let\process@line\@undefined
4741 \let\process@synonym\@undefined
4742 \let\process@language\@undefined
4743 \let\bbl@get@enc\@undefined
4744 \let\bbl@hyph@enc\@undefined
4745 \let\bbl@tempa\@undefined
4746 \let\bbl@hook@loadkernel\@undefined
4747 \let\bbl@hook@everylanguage\@undefined
```



```

4748 \let\bbl@hook@loadpatterns\@undefined
4749 \let\bbl@hook@loadexceptions\@undefined
4750 \end{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`).

```

4751 <<*More package options>> ≡
4752 \chardef\bbl@bidimode\z@
4753 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4754 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4755 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4756 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4757 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4758 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4759 <</More package options>>

```

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

```

4760 <<*Font selection>> ≡
4761 \bbl@trace{Font handling with fontspec}
4762 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4763 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4764 \DisableBabelHook{babel-fontspec}
4765 \onlypreamble\babelfont
4766 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4767   \ifx\fontspec\@undefined
4768     \usepackage{fontspec}%
4769     \fi
4770     \EnableBabelHook{babel-fontspec}%
4771     \edef\bbl@tempa{#1}%
4772     \def\bbl@tempb{#2}% Used by \bbl@bblfont
4773     \bbl@bblfont}
4774 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4775   \bbl@ifunset{\bbl@tempb family}%
4776     {\bbl@providefam{\bbl@tempb}}%
4777     {}%
4778   % For the default font, just in case:
4779   \bbl@ifunset{\bbl@sys@language}{\bbl@provide@sys{\language}}{}%
4780   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4781     {\bbl@csarg\edef{\bbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
4782     \bbl@exp{%
4783       \let<bbl@bbl@tempb dflt@\language>\<bbl@bbl@tempb dflt@>%
4784       \bbl@font@set<bbl@bbl@tempb dflt@\language>%
4785       \<bbl@tempb default>\<bbl@tempb family>}}%
4786     {\bbl@foreach\bbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4787       \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:

```

4788 \def\bbl@providefam#1{%
4789   \bbl@exp{%
4790     \newcommand<#1default>{}% Just define it
4791     \bbl@add@list{\bbl@font@fams{#1}}%
4792     \NewHook{#1family}%
4793     \DeclareRobustCommand<#1family>{%
4794       \not@math@alphabet<#1family>\relax
4795       % \prepare@family@series@update{#1}<#1default>% TODO. Fails

```

```

4796     \\fontfamily\<#1default>%
4797     \\UseHook{#1family}%
4798     \\selectfont}%
4799     \\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.

```

4800 \def\bbl@nostdfont#1{%
4801   \bbl@ifunset{bbl@WFF@f@family}%
4802   {\bbl@csarg\gdef{WFF@f@family}{}}% Flag, to avoid dupl warns
4803   \bbl@infowarn{The current font is not a babel standard family:\\%
4804     #1%
4805     \fontname\font\\%
4806     There is nothing intrinsically wrong with this warning, and\\%
4807     you can ignore it altogether if you do not need these\\%
4808     families. But if they are used in the document, you should be\\%
4809     aware 'babel' will not set Script and Language for them, so\\%
4810     you may consider defining a new family with \string\babelfont.\\%
4811     See the manual for further details about \string\babelfont.\\%
4812     Reported}}
4813   {}}%
4814 \gdef\bbl@switchfont{%
4815   \bbl@ifunset{bbl@lsys@language}{\bbl@provide@lsys{language}}}%
4816   \bbl@exp{ e.g., Arabic -> arabic
4817     \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}%
4818   \bbl@foreach\bbl@font@fams{%
4819     \bbl@ifunset{bbl@##1dflt@language}% (1) language?
4820     {\bbl@ifunset{bbl@##1dflt*%bbl@tempa}% (2) from script?
4821       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4822         {}}% 123=F - nothing!
4823       {\bbl@exp{ 3=T - from generic
4824         \global\let\bbl@##1dflt@language}%
4825         \<bbl@##1dflt@>}}}%
4826       {\bbl@exp{ 2=T - from script
4827         \global\let\bbl@##1dflt@language}%
4828         \<bbl@##1dflt*%bbl@tempa>}}}%
4829     {}}% 1=T - language, already defined
4830   \def\bbl@tempa{\bbl@nostdfont{}}%
4831   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4832     \bbl@ifunset{bbl@##1dflt@language}%
4833     {\bbl@cs{famrst@##1}%
4834     \global\bbl@csarg\let{famrst@##1}\relax}%
4835     {\bbl@exp{ order is relevant.
4836       \\bbl@add\\originalTeX%
4837       \\bbl@font@rst{\bbl@cl{##1dflt}}%
4838       \<##1default>\<##1family>{##1}}%
4839       \\bbl@font@set{\<bbl@##1dflt@language>% the main part!
4840       \<##1default>\<##1family>}}}%
4841   \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.

```

4842 \ifx\fontfamily\undefined\else % if latex
4843 \ifcase\bbl@engine % if pdftex
4844 \let\bbl@cckckstdfonts\relax
4845 \else
4846 \def\bbl@cckckstdfonts{%
4847   \begingroup
4848   \global\let\bbl@cckckstdfonts\relax
4849   \let\bbl@tempa\empty
4850   \bbl@foreach\bbl@font@fams{%
4851     \bbl@ifunset{bbl@##1dflt@}%
4852     {\@nameuse{##1family}%
4853     \bbl@csarg\gdef{WFF@f@family}{}}% Flag

```

```

4854      \bbl@exp{\bbl@add\bbl@tempa{* \<#1family>= \f@family\\}%
4855      \space\space\fontname\font\\}%
4856      \bbl@csarg\xdef{##1dflt@}{\f@family}%
4857      \expandafter\xdef\csname ##1default\endcsname{\f@family}%
4858      {}}%
4859      \ifx\bbl@tempa\empty\else
4860      \bbl@infowarn{The following font families will use the default\\%
4861      settings for all or some languages:\\%
4862      \bbl@tempa
4863      There is nothing intrinsically wrong with it, but\\%
4864      'babel' will no set Script and Language, which could\\%
4865      be relevant in some languages. If your document uses\\%
4866      these families, consider redefining them with \string\babelfont.\\%
4867      Reported}%
4868      \fi
4869      \endgroup}
4870 \fi
4871 \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, L^AT_EX 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*).

```

4872 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4873   \bbl@xin@{<>}{#1}%
4874   \ifin@
4875     \bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4876   \fi
4877   \bbl@exp{%
4878     \def\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}
4879     \bbl@ifsamestring{#2}{\f@family}%
4880     {\#3%
4881       \bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}%
4882       \let\bbl@tempa\relax}%
4883     {}%

```

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).

```

4884 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4885   \let\bbl@tempe\bbl@mapselect
4886   \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
4887   \bbl@exp{\bbl@replace\bbl@tempb{\bbl@stripslash\family/}}%
4888   \let\bbl@mapselect\relax
4889   \let\bbl@temp@fam#4% e.g., '\rmfamily', to be restored below
4890   \let#4\empty % Make sure \renewfontfamily is valid
4891   \bbl@set@renderer
4892   \bbl@exp{%
4893     \let\bbl@temp@pfam<\bbl@stripslash#4\space> e.g., '\rmfamily '
4894     \<keys_if_exist:nnf>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4895     {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4896     \<keys_if_exist:nnf>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4897     {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4898     \renewfontfamily\#4%
4899     [\bbl@cl{sys},% xetex removes unknown features :-(
4900     \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi

```

```

4901      #2]}{#3}% i.e., \bbl@exp{...}{#3}
4902 \bbl@unset@renderer
4903 \begingroup
4904      #4%
4905      \xdef#1{\f@family}%      e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4906 \endgroup
4907 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4908 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4909 \ifin@
4910 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4911 \fi
4912 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4913 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4914 \ifin@
4915 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4916 \fi
4917 \let#4\bbl@temp@fam
4918 \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4919 \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.

```

4920 \def\bbl@font@rst#1#2#3#4{%
4921 \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.

```

4922 \def\bbl@font@fams{rm,sf,tt}
4923 <</Font selection>>

```

10. Hooks for XeTeX and LuaTeX

10.1. XeTeX

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

Now, the code.

```

4924 <*>xetex<
4925 \def\BabelStringsDefault{unicode}
4926 \let\xebbl@stop\relax
4927 \AddBabelHook{xetex}{encodedcommands}{%
4928 \def\bbl@tempa{#1}%
4929 \ifx\bbl@tempa\@empty
4930 \XeTeXinputencoding"bytes"%
4931 \else
4932 \XeTeXinputencoding"#1"%
4933 \fi
4934 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4935 \AddBabelHook{xetex}{stopcommands}{%
4936 \xebbl@stop
4937 \let\xebbl@stop\relax}
4938 \def\bbl@input@classes{% Used in CJK intraspaces
4939 \input{load-unicode-xetex-classes.tex}%
4940 \let\bbl@input@classes\relax}
4941 \def\bbl@intraspace#1 #2 #3\@{
4942 \bbl@ccarg\gdef{xeisp@\languagename}%
4943 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4944 \def\bbl@intrapenalty#1\@{
4945 \bbl@ccarg\gdef{xeipn@\languagename}%
4946 {\XeTeXlinebreakpenalty #1\relax}}
4947 \def\bbl@provide@intraspace{%
4948 \bbl@xin@{/s}{\bbl@cl{lbrk}}%

```

```

4949 \ifin@else\bbl@xin@{/c}/{/bbl@c{l{lnbrk}}}\fi
4950 \ifin@
4951 \bbl@ifunset{bbl@intsp@language\language}\fi
4952 {\expandafter\ifx\csname bbl@intsp@language\endcsname\empty\else
4953 \ifx\bbl@KVP@intraspace\@nnil
4954 \bbl@exp{%
4955 \\\bbl@intraspace\bbl@c{l{intsp}}\@%
4956 \fi
4957 \ifx\bbl@KVP@intrapenalty\@nnil
4958 \bbl@intrapenalty0\@%
4959 \fi
4960 \fi
4961 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4962 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@%
4963 \fi
4964 \ifx\bbl@KVP@intrapenalty\@nnil\else
4965 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@%
4966 \fi
4967 \bbl@exp{%
4968 \\\bbl@add<extras\language>{%
4969 \XeTeXlinebreaklocale "\bbl@c{l{tbc}}"%
4970 \<bbl@xeisp@language>%
4971 \<bbl@xeipn@language>%
4972 \\\bbl@tglobal\<extras\language>%
4973 \\\bbl@add<noextras\language>{%
4974 \XeTeXlinebreaklocale ""}%
4975 \\\bbl@tglobal\<noextras\language>%
4976 \ifx\bbl@ispace\@undefined
4977 \gdef\bbl@ispace{\bbl@c{l{xeisp}}}%
4978 \ifx\AtBeginDocument\@notprerr
4979 \expandafter\@secondoftwo % to execute right now
4980 \fi
4981 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4982 \fi}%
4983 \fi}
4984 \ifx\DisableBabelHook\@undefined\endinput\fi
4985 \let\bbl@set@renderer\relax
4986 \let\bbl@unset@renderer\relax
4987 <@Font selection@>
4988 \def\bbl@provide@extra#1{}

```

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

```

4989 \def\bbl@xenohyph@d{%
4990 \bbl@ifset{bbl@prehc@language}\fi
4991 {\ifnum\hyphenchar\font=\defaultthyphenchar
4992 \iffontchar\font\bbl@c{l{prehc}}\relax
4993 \hyphenchar\font\bbl@c{l{prehc}}\relax
4994 \else\iffontchar\font"200B
4995 \hyphenchar\font"200B
4996 \else
4997 \bbl@warning
4998 {Neither 0 nor ZERO WIDTH SPACE are available\\%
4999 in the current font, and therefore the hyphen\\%
5000 will be printed. Try changing the fontspec's\\%
5001 'HyphenChar' to another value, but be aware\\%
5002 this setting is not safe (see the manual).\\%
5003 Reported}%
5004 \hyphenchar\font\defaultthyphenchar
5005 \fi\fi
5006 \fi}%
5007 {\hyphenchar\font\defaultthyphenchar}}

```

10.2. Support for interchar

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

```
5008 \ifnum\xe@alloc@intercharclass<\thr@@
5009 \xe@alloc@intercharclass\thr@@
5010 \fi
5011 \chardef\bbl@xe@class@default=\z@
5012 \chardef\bbl@xe@class@cjkideogram=\@ne
5013 \chardef\bbl@xe@class@cjkleftpunctuation=\tw@
5014 \chardef\bbl@xe@class@cjkrightpunctuation=\thr@@
5015 \chardef\bbl@xe@class@boundary=4095
5016 \chardef\bbl@xe@class@ignore=4096
```

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

```
5017 \AddBabelHook{babel-interchar}{beforeextras}{%
5018 \nameuse{bbl@xechars@language}
5019 \DisableBabelHook{babel-interchar}
5020 \protected\def\bbl@charclass#1{%
5021 \ifnum\count@<\z@
5022 \count@-\count@
5023 \loop
5024 \bbl@exp{%
5025 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}%
5026 \XeTeXcharclass\count@ \bbl@tempc
5027 \ifnum\count@<`#1\relax
5028 \advance\count@\@ne
5029 \repeat
5030 \else
5031 \babel@savevariable{\XeTeXcharclass`#1}%
5032 \XeTeXcharclass`#1 \bbl@tempc
5033 \fi
5034 \count@`#1\relax}
```

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

```
5035 \newcommand\bbl@ifinterchar[1]{%
5036 \let\bbl@tempa@gobble % Assume to ignore
5037 \edef\bbl@tempb{\zap@space#1 \@empty}%
5038 \ifx\bbl@KVP@interchar@nnil\else
5039 \bbl@replace\bbl@KVP@interchar{ }{,}%
5040 \bbl@foreach\bbl@tempb{%
5041 \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5042 \ifin@
5043 \let\bbl@tempa@firstofone
5044 \fi}%
5045 \fi
5046 \bbl@tempa}
5047 \newcommand\IfBabelIntercharT[2]{%
5048 \bbl@carg\bbl@add{bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5049 \newcommand\babelcharclass[3]{%
5050 \EnableBabelHook{babel-interchar}%
5051 \bbl@csarg\newXeTeXintercharclass{xe@class@#2@#1}%
5052 \def\bbl@tempb##1{%
5053 \ifx##1\@empty\else
5054 \ifx##1-%
5055 \bbl@upto
```

```

5056     \else
5057         \bbl@charclass{%
5058             \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5059         \fi
5060         \expandafter\bbl@tempb
5061     \fi}%
5062 \bbl@ifunset{bbl@xechars@#1}%
5063     {\toks@{%
5064         \babel@savevariable\XeTeXinterchartokenstate
5065         \XeTeXinterchartokenstate\@ne
5066     }}%
5067     {\toks@\expandafter\expandafter\expandafter{%
5068         \csname bbl@xechars@#1\endcsname}}%
5069 \bbl@csarg\edef{xechars@#1}{%
5070     \the\toks@
5071     \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
5072     \bbl@tempb#3\@empty}}
5073 \protected\def\bbl@usingxeclasse#1{\count@\z@ \let\bbl@tempc#1}
5074 \protected\def\bbl@upto{%
5075     \ifnum\count@>\z@
5076         \advance\count@\@ne
5077         \count@-\count@
5078     \else\ifnum\count@=\z@
5079         \bbl@charclass{-}%
5080     \else
5081         \bbl@error{double-hyphens-class}{\count@}{\count@}%
5082     \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>`.

```

5083 \def\bbl@ignoreinterchar{%
5084     \ifnum\language=\l@nohyphenation
5085         \expandafter\@gobble
5086     \else
5087         \expandafter\@firstofone
5088     \fi}
5089 \newcommand\babelinterchar[5][]{%
5090     \let\bbl@kv@label\@empty
5091     \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5092     \namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5093     {\bbl@ignoreinterchar{#5}}%
5094     \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5095     \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5096         \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5097             \XeTeXinterchartoks
5098             \@nameuse{bbl@xeclasse@\bbl@tempa @#2}
5099             \bbl@ifunset{bbl@xeclasse@\bbl@tempa @#2}{\bbl@tempc@#2} %
5100             \@nameuse{bbl@xeclasse@\bbl@tempb @#2}
5101             \bbl@ifunset{bbl@xeclasse@\bbl@tempb @#2}{\bbl@tempc@#2} %
5102             = \expandafter{%
5103                 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5104                 \csname\zap@space bbl@xeinter@\bbl@kv@label
5105                     @#3@#4@#2 \@empty\endcsname}}}}
5106 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5107     \bbl@ifunset{bbl@ic@#1@language}%
5108     {\bbl@error{unknown-interchar}{#1}{\count@}}%
5109     {\bbl@csarg\let{ic@#1@language}\@firstofone}}
5110 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5111     \bbl@ifunset{bbl@ic@#1@language}%
5112     {\bbl@error{unknown-interchar-b}{#1}{\count@}}%
5113     {\bbl@csarg\let{ic@#1@language}\@gobble}}
5114 </xetex>

```

10.3. Layout

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

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

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

```
5115 < *xetex | texxet >
5116 \providecommand\bbl@provide@intraspace{}
5117 \bbl@trace{Redefinitions for bidi layout}

    Finish here if there is no layout.

5118 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5119 \IfBabelLayout{nopars}
5120 {}
5121 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5122 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5123 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5124 \ifnum\bbl@bidimode>\z@
5125 \IfBabelLayout{pars}
5126 {\def\@hangfrom#1{%
5127   \setbox\@tempboxa\hbox{{#1}}%
5128   \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5129   \noindent\box\@tempboxa}
5130 \def\raggedright{%
5131   \let\\\@centercr
5132   \bbl@startskip\z@skip
5133   \@rightskip\@flushglue
5134   \bbl@endskip\@rightskip
5135   \parindent\z@
5136   \parfillskip\bbl@startskip}
5137 \def\raggedleft{%
5138   \let\\\@centercr
5139   \bbl@startskip\@flushglue
5140   \bbl@endskip\z@skip
5141   \parindent\z@
5142   \parfillskip\bbl@endskip}}
5143 {}
5144 \fi
5145 \IfBabelLayout{lists}
5146 {\bbl@sreplace\list
5147   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5148   \def\bbl@listleftmargin{%
5149     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5150   \ifcase\bbl@engine
5151     \def\labelenumii{}\theenumii{}% pdfTeX doesn't reverse ()
5152     \def\p@enumiii{\p@enumii}\theenumii{}%
5153   \fi
5154   \bbl@sreplace\@verbatim
5155     {\leftskip\@totalleftmargin}%
5156     {\bbl@startskip\textwidth
5157       \advance\bbl@startskip-\linewidth}%
5158   \bbl@sreplace\@verbatim
5159     {\rightskip\z@skip}%
5160     {\bbl@endskip\z@skip}}%
5161 {}
5162 \IfBabelLayout{contents}
5163 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5164   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5165 {}
5166 \IfBabelLayout{columns}
```



```

5167 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5168 \def\bbl@outputbox#1{%
5169   \hb@xt@\textwidth{%
5170     \hskip\columnwidth
5171     \hfil
5172     {\normalcolor\vrule \@width\columnseprule}%
5173     \hfil
5174     \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5175     \hskip-\textwidth
5176     \hb@xt@\columnwidth{\box\@outputbox \hss}%
5177     \hskip\columnsep
5178     \hskip\columnwidth}}}%
5179 {}

```

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.

```

5180 \IfBabelLayout{counters*}%
5181 {\bbl@add\bbl@opt@layout{.counters.}%
5182 \AddToHook{shipout/before}{%
5183   \let\bbl@tempa\babelsublr
5184   \let\babelsublr\@firstofone
5185   \let\bbl@save@thepage\thepage
5186   \protected@edef\thepage{\thepage}%
5187   \let\babelsublr\bbl@tempa}%
5188 \AddToHook{shipout/after}{%
5189   \let\thepage\bbl@save@thepage}}{}
5190 \IfBabelLayout{counters}%
5191 {\let\bbl@latinarabic=\@arabic
5192 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
5193 \let\bbl@asciroman=\@roman
5194 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5195 \let\bbl@asciiRoman=\@Roman
5196 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5197 \fi % end if layout
5198 \xetex | texxet

```

10.4. 8-bit TeX

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

```

5199 \xetex | texxet
5200 \def\bbl@provide@extra#1{%
5201   % == auto-select encoding ==
5202   \ifx\bbl@encoding@select\off\@empty\else
5203     \bbl@ifunset{\bbl@encoding@#1}%
5204     {\def\@elt##1{,##1,}%
5205     \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5206     \count@\z@
5207     \bbl@foreach\bbl@tempe{%
5208       \def\bbl@tempd{##1}% Save last declared
5209       \advance\count@\@one}%
5210     \ifnum\count@>\@one % (1)
5211       \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5212       \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5213       \bbl@replace\bbl@tempa{ },}%
5214       \global\bbl@csarg\let{encoding@#1}\@empty
5215       \bbl@xin@{\bbl@tempd,}{,\bbl@tempa,}%
5216       \ifin@ \else % if main encoding included in ini, do nothing
5217         \let\bbl@tempb\relax
5218         \bbl@foreach\bbl@tempa{%
5219           \ifx\bbl@tempb\relax
5220             \bbl@xin@{,##1,}{,\bbl@tempb,}%
5221             \ifin@\def\bbl@tempb{##1}\fi

```

```

5222         \fi}%
5223     \ifx\bbbl@tempb\relax\else
5224         \bbbl@exp{%
5225             \global\<bbbl@add>\<bbbl@preextras@#1>{\<bbbl@encoding@#1>}%
5226             \gdef\<bbbl@encoding@#1>{%
5227                 \\babel@save\\f@encoding
5228                 \\bbbl@add\\originalTeX{\\selectfont}%
5229                 \\fontencoding{\bbbl@tempb}%
5230                 \\selectfont}}%
5231         \fi
5232     \fi
5233     \fi}%
5234     }%
5235     \fi}
5236 \</texset>

```

10.5. LuaTeX

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

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

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

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

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

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

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

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

```

5237 \<!*luatex>
5238 \directlua{ Babel = Babel or {} } % DL2
5239 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5240 \bbbl@trace{Read language.dat}
5241 \ifx\bbbl@readstream\undefined
5242     \csname newread\endcsname\bbbl@readstream
5243 \fi
5244 \begingroup
5245     \toks@{}
5246     \count@ \z@ % 0=start, 1=0th, 2=normal
5247     \def\bbbl@process@line#1#2 #3 #4 {%
5248         \ifx=#1%
5249             \bbbl@process@synonym#2}%

```

```

5250 \else
5251 \bbl@process@language{#1#2}{#3}{#4}%
5252 \fi
5253 \ignorespaces}
5254 \def\bbl@manylang{%
5255 \ifnum\bbl@last>\@ne
5256 \bbl@info{Non-standard hyphenation setup}%
5257 \fi
5258 \let\bbl@manylang\relax}
5259 \def\bbl@process@language#1#2#3{%
5260 \ifcase\count@
5261 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5262 \or
5263 \count@\tw@
5264 \fi
5265 \ifnum\count@=\tw@
5266 \expandafter\addlanguage\csname l@#1\endcsname
5267 \language\allocationnumber
5268 \chardef\bbl@last\allocationnumber
5269 \bbl@manylang
5270 \let\bbl@elt\relax
5271 \xdef\bbl@languages{%
5272 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5273 \fi
5274 \the\toks@
5275 \toks@{}}
5276 \def\bbl@process@synonym@aux#1#2{%
5277 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5278 \let\bbl@elt\relax
5279 \xdef\bbl@languages{%
5280 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5281 \def\bbl@process@synonym#1{%
5282 \ifcase\count@
5283 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5284 \or
5285 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{%
5286 \else
5287 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5288 \fi}
5289 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
5290 \chardef\l@english\z@
5291 \chardef\l@USenglish\z@
5292 \chardef\bbl@last\z@
5293 \global\@namedef{bbl@hyphendata@0}{{\hyphen.tex}}
5294 \gdef\bbl@languages{%
5295 \bbl@elt{english}{0}{\hyphen.tex}}%
5296 \bbl@elt{USenglish}{0}{}}
5297 \else
5298 \global\let\bbl@languages@format\bbl@languages
5299 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5300 \ifnum#2>\z@
5301 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5302 \fi}%
5303 \xdef\bbl@languages{\bbl@languages}%
5304 \fi
5305 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5306 \bbl@languages
5307 \openin\bbl@readstream=language.dat
5308 \ifeof\bbl@readstream
5309 \bbl@warning{I couldn't find language.dat. No additional\\%
5310 patterns loaded. Reported}%
5311 \else
5312 \loop

```

```

5313     \endlinechar\m@ne
5314     \read\bbbl@readstream to \bbbl@line
5315     \endlinechar`\^^M
5316     \if T\ifeof\bbbl@readstream F\fi T\relax
5317     \ifx\bbbl@line\@empty\else
5318         \edef\bbbl@line{\bbbl@line\space\space\space}%
5319         \expandafter\bbbl@process@line\bbbl@line\relax
5320     \fi
5321 \repeat
5322 \fi
5323 \closein\bbbl@readstream
5324 \endgroup
5325 \bbbl@trace{Macros for reading patterns files}
5326 \def\bbbl@get@enc#1:#2:#3\@@@{\def\bbbl@hyph@enc{#2}}
5327 \ifx\babelcatcodetablenum\@undefined
5328     \ifx\newcatcodetable\@undefined
5329         \def\babelcatcodetablenum{5211}
5330         \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5331     \else
5332         \newcatcodetable\babelcatcodetablenum
5333         \newcatcodetable\bbbl@pattcodes
5334     \fi
5335 \else
5336     \def\bbbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5337 \fi
5338 \def\bbbl@luapatterns#1#2{%
5339     \bbbl@get@enc#1:.\@@@
5340     \setbox\z@\hbox\bgroup
5341         \beginingroup
5342             \savecatcodetable\babelcatcodetablenum\relax
5343             \initcatcodetable\bbbl@pattcodes\relax
5344             \catcodetable\bbbl@pattcodes\relax
5345             \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5346             \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
5347             \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5348             \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5349             \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5350             \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5351             \input #1\relax
5352         \catcodetable\babelcatcodetablenum\relax
5353     \endgroup
5354     \def\bbbl@tempa{#2}%
5355     \ifx\bbbl@tempa\@empty\else
5356         \input #2\relax
5357     \fi
5358 \egroup}%
5359 \def\bbbl@patterns@lua#1{%
5360     \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5361         \csname l@#1\endcsname
5362         \edef\bbbl@tempa{#1}%
5363     \else
5364         \csname l@#1:\f@encoding\endcsname
5365         \edef\bbbl@tempa{#1:\f@encoding}%
5366     \fi\relax
5367     \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5368     \@ifundefined{bbbl@hyphendata@the\language}%
5369     {\def\bbbl@elt##1##2###3###4{%
5370         \ifnum##2=\csname l@bbbl@tempa\endcsname % #2=spanish, dutch:OT1...
5371         \def\bbbl@tempb{##3}%
5372         \ifx\bbbl@tempb\@empty\else % if not a synonymous
5373             \def\bbbl@tempc{{##3}{##4}}%
5374         \fi
5375         \bbbl@csarg\xdef{hyphendata@##2}{\bbbl@tempc}%

```

```

5376     \fi}%
5377     \bbl@languages
5378     \ifundefined{bbl@hyphendata@the\language}%
5379     {\bbl@info{No hyphenation patterns were set for\%
5380       language 'bbl@tempa'. Reported}}%
5381     {\expandafter\expandafter\expandafter\bbl@luapatterns
5382       \csname bbl@hyphendata@the\language\endcsname}}{}
5383 \endinput\fi

```

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

```

5384 \ifx\DisableBabelHook\@undefined
5385   \AddBabelHook{luatex}{everylanguage}{%
5386     \def\process@language##1##2##3{%
5387       \def\process@line####1####2 ####3 ####4 {}}}
5388   \AddBabelHook{luatex}{loadpatterns}{%
5389     \input #1\relax
5390     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5391       {#{1}}}}
5392   \AddBabelHook{luatex}{loadexceptions}{%
5393     \input #1\relax
5394     \def\bbl@tempb##1##2{#{1}}}%
5395     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5396       {\expandafter\expandafter\expandafter\bbl@tempb
5397         \csname bbl@hyphendata@the\language\endcsname}}
5398 \endinput\fi

```

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

```

5399 \begingroup
5400 \catcode`\%=12
5401 \catcode`\'=12
5402 \catcode`\=12
5403 \catcode`\:=12
5404 \directlua{
5405   Babel.locale_props = Babel.locale_props or {}
5406   function Babel.lua_error(e, a)
5407     tex.print([[noexpand\csname bbl@error\endcsname]] ..
5408       e .. '{' .. (a or '') .. '}{}')
5409   end
5410
5411   function Babel.bytes(line)
5412     return line:gsub(".",
5413       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5414   end
5415
5416   function Babel.priority_in_callback(name,description)
5417     for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5418       if v == description then return i end
5419     end
5420     return false
5421   end
5422
5423   function Babel.begin_process_input()
5424     if luatexbase and luatexbase.add_to_callback then
5425       luatexbase.add_to_callback('process_input_buffer',
5426         Babel.bytes, 'Babel.bytes')
5427     else
5428       Babel.callback = callback.find('process_input_buffer')
5429       callback.register('process_input_buffer',Babel.bytes)
5430     end
5431   end
5432   function Babel.end_process_input ()
5433     if luatexbase and luatexbase.remove_from_callback then
5434       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')

```

```

5435     else
5436         callback.register('process_input_buffer',Babel.callback)
5437     end
5438 end
5439
5440 function Babel.str_to_nodes(fn, matches, base)
5441     local n, head, last
5442     if fn == nil then return nil end
5443     for s in string.utfvalues(fn(matches)) do
5444         if base.id == 7 then
5445             base = base.replace
5446         end
5447         n = node.copy(base)
5448         n.char = s
5449         if not head then
5450             head = n
5451         else
5452             last.next = n
5453         end
5454         last = n
5455     end
5456     return head
5457 end
5458
5459 Babel.linebreaking = Babel.linebreaking or {}
5460 Babel.linebreaking.before = {}
5461 Babel.linebreaking.after = {}
5462 Babel.locale = {}
5463 function Babel.linebreaking.add_before(func, pos)
5464     tex.print([[\\noexpand\\csname bbl@luahyphenate\\endcsname]])
5465     if pos == nil then
5466         table.insert(Babel.linebreaking.before, func)
5467     else
5468         table.insert(Babel.linebreaking.before, pos, func)
5469     end
5470 end
5471 function Babel.linebreaking.add_after(func)
5472     tex.print([[\\noexpand\\csname bbl@luahyphenate\\endcsname]])
5473     table.insert(Babel.linebreaking.after, func)
5474 end
5475
5476 function Babel.addpatterns(pp, lg)
5477     local lg = lang.new(lg)
5478     local pats = lang.patterns(lg) or ''
5479     lang.clear_patterns(lg)
5480     for p in pp:gmatch('[^%s]+') do
5481         ss = ''
5482         for i in string.utfcharacters(p:gsub('%d', '')) do
5483             ss = ss .. '%d?' .. i
5484         end
5485         ss = ss:gsub('^%d%?%.','%.') .. '%d?'
5486         ss = ss:gsub('%.%d%?$', '%%.')
5487         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5488         if n == 0 then
5489             tex.sprint(
5490                 [[\\string\\csname\\space bbl@info\\endcsname{New pattern: }]]
5491                 .. p .. [[{ }]])
5492             pats = pats .. ' ' .. p
5493         else
5494             tex.sprint(
5495                 [[\\string\\csname\\space bbl@info\\endcsname{Renew pattern: }]]
5496                 .. p .. [[{ }]])
5497         end
5498     end

```

```

5498     end
5499     lang.patterns(lg, pats)
5500 end
5501
5502 Babel.characters = Babel.characters or {}
5503 Babel.ranges = Babel.ranges or {}
5504 function Babel.hlist_has_bidi(head)
5505     local has_bidi = false
5506     local ranges = Babel.ranges
5507     for item in node.traverse(head) do
5508         if item.id == node.id'glyph' then
5509             local itemchar = item.char
5510             local chardata = Babel.characters[itemchar]
5511             local dir = chardata and chardata.d or nil
5512             if not dir then
5513                 for nn, et in ipairs(ranges) do
5514                     if itemchar < et[1] then
5515                         break
5516                     elseif itemchar <= et[2] then
5517                         dir = et[3]
5518                         break
5519                     end
5520                 end
5521             end
5522             if dir and (dir == 'al' or dir == 'r') then
5523                 has_bidi = true
5524             end
5525         end
5526     end
5527     return has_bidi
5528 end
5529 function Babel.set_chranges_b (script, chrng)
5530     if chrng == '' then return end
5531     texio.write('Replacing ' .. script .. ' script ranges')
5532     Babel.script_blocks[script] = {}
5533     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5534         table.insert(
5535             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5536     end
5537 end
5538
5539 function Babel.discard_sublr(str)
5540     if str:find( [[\string\indexentry]] ) and
5541        str:find( [[\string\babelsublr]] ) then
5542         str = str:gsub( [[\string\babelsublr%*{b}{}]] ,
5543             function(m) return m:sub(2,-2) end )
5544     end
5545     return str
5546 end
5547 }
5548 \endgroup
5549 \ifx\newattribute\@undefined\else % Test for plain
5550     \newattribute\bbl@attr@locale % DL4
5551     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5552     \AddBabelHook{luatex}{beforeextras}{%
5553         \setattribute\bbl@attr@locale\localeid}
5554 \fi
5555 %
5556 \def\BabelStringsDefault{unicode}
5557 \let\luabbbl@stop\relax
5558 \AddBabelHook{luatex}{encodedcommands}{%
5559     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5560     \ifx\bbl@tempa\bbl@tempb\else

```

```

5561 \directlua{Babel.begin_process_input()}%
5562 \def\luabbl@stop{%
5563 \directlua{Babel.end_process_input()}}%
5564 \fi}%
5565 \AddBabelHook{luatex}{stopcommands}{%
5566 \luabbl@stop
5567 \let\luabbl@stop\relax}
5568 %
5569 \AddBabelHook{luatex}{patterns}{%
5570 \ifundefined{bbl@hyphendata@the\language}%
5571 {\def\bbl@elt##1##2##3##4{%
5572 \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5573 \def\bbl@tempb{##3}%
5574 \ifx\bbl@tempb\empty\else % if not a synonymous
5575 \def\bbl@tempc{##3}{##4}}%
5576 \fi
5577 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5578 \fi}%
5579 \bbl@languages
5580 \ifundefined{bbl@hyphendata@the\language}%
5581 {\bbl@info{No hyphenation patterns were set for\%
5582 language '#2'. Reported}}%
5583 {\expandafter\expandafter\expandafter\bbl@luapatterns
5584 \csname bbl@hyphendata@the\language\endcsname}}}%
5585 \ifundefined{bbl@patterns@}{}%
5586 \begingroup
5587 \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5588 \ifin@else
5589 \ifx\bbl@patterns@\empty\else
5590 \directlua{ Babel.addpatterns(
5591 [[\bbl@patterns@]], \number\language) }%
5592 \fi
5593 \ifundefined{bbl@patterns@#1}%
5594 \empty
5595 {\directlua{ Babel.addpatterns(
5596 [[\space\csname bbl@patterns@#1\endcsname]],
5597 \number\language) }}%
5598 \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5599 \fi
5600 \endgroup}%
5601 \bbl@exp{%
5602 \bbl@ifunset{bbl@prehc@\languagename}{}%
5603 {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5604 {\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.

```

5605 \@onlypreamble\babelpatterns
5606 \AtEndOfPackage{%
5607 \newcommand\babelpatterns[2][\empty]{%
5608 \ifx\bbl@patterns@relax
5609 \let\bbl@patterns@\empty
5610 \fi
5611 \ifx\bbl@pttnlist\empty\else
5612 \bbl@warning{%
5613 You must not intermingle \string\selectlanguage\space and\%
5614 \string\babelpatterns\space or some patterns will not\%
5615 be taken into account. Reported}%
5616 \fi
5617 \ifx\empty#1%
5618 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5619 \else

```



```

5620 \edef\bbl@tempb{\zap@space#1 \@empty}%
5621 \bbl@for\bbl@tempa\bbl@tempb{%
5622 \bbl@fixname\bbl@tempa
5623 \bbl@iflanguage\bbl@tempa{%
5624 \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5625 \@ifundefined{bbl@patterns@\bbl@tempa}%
5626 \@empty
5627 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5628 #2}}}%
5629 \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.

```

5630 \def\bbl@intraspace#1 #2 #3\@{
5631 \directlua{
5632 Babel.intraspaces = Babel.intraspaces or {}
5633 Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5634 {b = #1, p = #2, m = #3}
5635 Babel.locale_props[\the\localeid].intraspace = %
5636 {b = #1, p = #2, m = #3}
5637 }}
5638 \def\bbl@intrapenalty#1\@{
5639 \directlua{
5640 Babel.intrapenalties = Babel.intrapenalties or {}
5641 Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5642 Babel.locale_props[\the\localeid].intrapenalty = #1
5643 }}
5644 \begingroup
5645 \catcode`\%=12
5646 \catcode`\&=14
5647 \catcode`\'=12
5648 \catcode`\-=12
5649 \gdef\bbl@seaintraspace{&
5650 \let\bbl@seaintraspace\relax
5651 \directlua{
5652 Babel.sea_enabled = true
5653 Babel.sea_ranges = Babel.sea_ranges or {}
5654 function Babel.set_chranges (script, chrng)
5655 local c = 0
5656 for s, e in string.gmatch(chrng..' ', '(.-%.%.(.%)s') do
5657 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5658 c = c + 1
5659 end
5660 end
5661 function Babel.sea_disc_to_space (head)
5662 local sea_ranges = Babel.sea_ranges
5663 local last_char = nil
5664 local quad = 655360 & 10 pt = 655360 = 10 * 65536
5665 for item in node.traverse(head) do
5666 local i = item.id
5667 if i == node.id'glyph' then
5668 last_char = item
5669 elseif i == 7 and item.subtype == 3 and last_char
5670 and last_char.char > 0x0C99 then
5671 quad = font.getfont(last_char.font).size
5672 for lg, rg in pairs(sea_ranges) do
5673 if last_char.char > rg[1] and last_char.char < rg[2] then
5674 lg = lg:sub(1, 4) & Remove trailing number of, e.g., Cyril
5675 local intraspace = Babel.intraspaces[lg]

```

```

5676         local intrapenalty = Babel.intrapenalties[lg]
5677         local n
5678         if intrapenalty ~= 0 then
5679             n = node.new(14, 0)      &% penalty
5680             n.penalty = intrapenalty
5681             node.insert_before(head, item, n)
5682         end
5683         n = node.new(12, 13)        &% (glue, spaceskip)
5684         node.setglue(n, intraspace.b * quad,
5685                       intraspace.p * quad,
5686                       intraspace.m * quad)
5687         node.insert_before(head, item, n)
5688         node.remove(head, item)
5689     end
5690 end
5691 end
5692 end
5693 end
5694 }&
5695 \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.

```

5696 \catcode`\%=14
5697 \gdef\bbl@cjkintraspacespace{%
5698   \let\bbl@cjkintraspacespace\relax
5699   \directlua{
5700     require('babel-data-cjk.lua')
5701     Babel.cjk_enabled = true
5702     function Babel.cjk_linebreak(head)
5703       local GLYPH = node.id'glyph'
5704       local last_char = nil
5705       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5706       local last_class = nil
5707       local last_lang = nil
5708       for item in node.traverse(head) do
5709         if item.id == GLYPH then
5710           local lang = item.lang
5711           local LOCALE = node.get_attribute(item,
5712             Babel.attr_locale)
5713           local props = Babel.locale_props[LOCALE] or {}
5714           local class = Babel.cjk_class[item.char].c
5715           if props.cjk_quotes and props.cjk_quotes[item.char] then
5716             class = props.cjk_quotes[item.char]
5717           end
5718           if class == 'cp' then class = 'cl' % )] as CL
5719           elseif class == 'id' then class = 'I'
5720           elseif class == 'cj' then class = 'I' % loose
5721           end
5722           local br = 0
5723           if class and last_class and Babel.cjk_breaks[last_class][class] then
5724             br = Babel.cjk_breaks[last_class][class]
5725           end
5726           if br == 1 and props.linebreak == 'c' and
5727             lang ~= \the\l@nohyphenation\space and
5728             last_lang ~= \the\l@nohyphenation then
5729             local intrapenalty = props.intrapenalty

```

```

5730         if intrapenalty ~= 0 then
5731             local n = node.new(14, 0)      % penalty
5732             n.penalty = intrapenalty
5733             node.insert_before(head, item, n)
5734         end
5735         local intraspace = props.intraspace
5736         local n = node.new(12, 13)        % (glue, spaceskip)
5737         node.setglue(n, intraspace.b * quad,
5738                     intraspace.p * quad,
5739                     intraspace.m * quad)
5740         node.insert_before(head, item, n)
5741     end
5742     if font.getfont(item.font) then
5743         quad = font.getfont(item.font).size
5744     end
5745     last_class = class
5746     last_lang = lang
5747     else % if penalty, glue or anything else
5748         last_class = nil
5749     end
5750 end
5751 lang.hyphenate(head)
5752 end
5753 }%
5754 \bbl@luahyphenate}
5755 \gdef\bbl@luahyphenate{%
5756 \let\bbl@luahyphenate\relax
5757 \directlua{
5758     luatexbase.add_to_callback('hyphenate',
5759     function (head, tail)
5760         if Babel.linebreaking.before then
5761             for k, func in ipairs(Babel.linebreaking.before) do
5762                 func(head)
5763             end
5764         end
5765         lang.hyphenate(head)
5766         if Babel.cjk_enabled then
5767             Babel.cjk_linebreak(head)
5768         end
5769         if Babel.linebreaking.after then
5770             for k, func in ipairs(Babel.linebreaking.after) do
5771                 func(head)
5772             end
5773         end
5774         if Babel.set_hboxed then
5775             Babel.set_hboxed(head)
5776         end
5777         if Babel.sea_enabled then
5778             Babel.sea_disc_to_space(head)
5779         end
5780     end,
5781     'Babel.hyphenate')
5782 }}
5783 \endgroup
5784 %
5785 \def\bbl@provide@intraspace{%
5786 \bbl@ifunset\bbl@intsp@\languagename}{}%
5787 {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5788 \bbl@xin@{/c}{/\bbl@cl{lnbrk}}}%
5789 \ifin@          % cjk
5790 \bbl@cjk_intraspace
5791 \directlua{
5792     Babel.locale_props = Babel.locale_props or {}

```

```

5793         Babel.locale_props[\the\localeid].linebreak = 'c'
5794     }%
5795     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\bbl@{}%
5796     \ifx\bbl@KVP@intrapenalty\@nnil
5797         \bbl@intrapenalty0\@
5798     \fi
5799 \else           % sea
5800     \bbl@seaintraspace
5801     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\bbl@{}%
5802     \directlua{
5803         Babel.sea_ranges = Babel.sea_ranges or {}
5804         Babel.set_chranges('\bbl@cl{sbcpr}',
5805             '\bbl@cl{chrng}')
5806     }%
5807     \ifx\bbl@KVP@intrapenalty\@nnil
5808         \bbl@intrapenalty0\@
5809     \fi
5810 \fi
5811 \fi
5812 \ifx\bbl@KVP@intrapenalty\@nnil\else
5813     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5814 \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`.

```

5815 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5816 \def\bblar@chars{%
5817     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5818     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5819     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5820 \def\bblar@elongated{%
5821     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5822     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5823     0649,064A}
5824 \begingroup
5825     \catcode\_ =11 \catcode\`:=11
5826     \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5827 \endgroup
5828 \gdef\bbl@arabicjust{%
5829     \let\bbl@arabicjust\relax
5830     \newattribute\bblar@kashida
5831     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5832     \bblar@kashida=\z@
5833     \bbl@patchfont{\bbl@parsejalt}}%
5834 \directlua{
5835     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5836     Babel.arabic.elong_map[\the\localeid] = {}
5837     luatexbase.add_to_callback('post_linebreak_filter',
5838         Babel.arabic.justify, 'Babel.arabic.justify')
5839     luatexbase.add_to_callback('hpack_filter',
5840         Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5841 }}%

```

Save both node lists to make replacement.

```

5842 \def\bblar@fetchjalt#1#2#3#4{%
5843     \bbl@exp{\bbl@foreach{#1}}{%
5844         \bbl@ifunset\bblar@JE@##1{%
5845             {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5846             {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}}%
5847     \directlua{%
5848         local last = nil

```

```

5849     for item in node.traverse(tex.box[0].head) do
5850         if item.id == node.id'glyph' and item.char > 0x600 and
5851             not (item.char == 0x200D) then
5852             last = item
5853         end
5854     end
5855     Babel.arabic.#3['##1#4'] = last.char
5856 }}

```

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

```

5857 \gdef\bbl@parsejalt{%
5858   \ifx\addfontfeature\undefined\else
5859     \bbl@xin{/e}/{\bbl@ccl{\lnbrk}}%
5860     \ifin@
5861       \directlua{%
5862         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5863           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5864           tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5865         end
5866       }%
5867     \fi
5868   \fi}
5869 \gdef\bbl@parsejalti{%
5870   \begingroup
5871     \let\bbl@parsejalt\relax % To avoid infinite loop
5872     \edef\bbl@tempb{\fontid\font}%
5873     \bblar@nofswarn
5874     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5875     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5876     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5877     \addfontfeature{RawFeature+=jalt}%
5878     % \@namedef{\bblar@JE@0643}{06AA}% todo: catch medial kaf
5879     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5880     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5881     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5882     \directlua{%
5883       for k, v in pairs(Babel.arabic.from) do
5884         if Babel.arabic.dest[k] and
5885             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5886           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5887             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5888         end
5889       end
5890     }%
5891   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5892 \begingroup
5893 \catcode`#=11
5894 \catcode`~=11
5895 \directlua{
5896
5897 Babel.arabic = Babel.arabic or {}
5898 Babel.arabic.from = {}
5899 Babel.arabic.dest = {}
5900 Babel.arabic.justify_factor = 0.95
5901 Babel.arabic.justify_enabled = true
5902 Babel.arabic.kashida_limit = -1
5903
5904 function Babel.arabic.justify(head)
5905   if not Babel.arabic.justify_enabled then return head end
5906   for line in node.traverse_id(node.id'hlist', head) do
5907     Babel.arabic.justify_hlist(head, line)

```

```

5908 end
5909 % In case the very first item is a line (eg, in \vbox):
5910 while head.prev do head = head.prev end
5911 return head
5912 end
5913
5914 function Babel.arabic.justify_hbox(head, gc, size, pack)
5915     local has_inf = false
5916     if Babel.arabic.justify_enabled and pack == 'exactly' then
5917         for n in node.traverse_id(12, head) do
5918             if n.stretch_order > 0 then has_inf = true end
5919         end
5920         if not has_inf then
5921             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5922         end
5923     end
5924     return head
5925 end
5926
5927 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5928     local d, new
5929     local k_list, k_item, pos_inline
5930     local width, width_new, full, k_curr, wt_pos, goal, shift
5931     local subst_done = false
5932     local elong_map = Babel.arabic.elong_map
5933     local cnt
5934     local last_line
5935     local GLYPH = node.id'glyph'
5936     local KASHIDA = Babel.attr_kashida
5937     local LOCALE = Babel.attr_locale
5938
5939     if line == nil then
5940         line = {}
5941         line.glue_sign = 1
5942         line.glue_order = 0
5943         line.head = head
5944         line.shift = 0
5945         line.width = size
5946     end
5947
5948     % Exclude last line. todo. But-- it discards one-word lines, too!
5949     % ? Look for glue = 12:15
5950     if (line.glue_sign == 1 and line.glue_order == 0) then
5951         elongs = {} % Stores elongated candidates of each line
5952         k_list = {} % And all letters with kashida
5953         pos_inline = 0 % Not yet used
5954
5955         for n in node.traverse_id(GLYPH, line.head) do
5956             pos_inline = pos_inline + 1 % To find where it is. Not used.
5957
5958             % Elongated glyphs
5959             if elong_map then
5960                 local locale = node.get_attribute(n, LOCALE)
5961                 if elong_map[locale] and elong_map[locale][n.font] and
5962                     elong_map[locale][n.font][n.char] then
5963                     table.insert(elongs, {node = n, locale = locale} )
5964                     node.set_attribute(n.prev, KASHIDA, 0)
5965                 end
5966             end
5967
5968             % Tatwil. First create a list of nodes marked with kashida. The
5969             % rest of nodes can be ignored. The list of used weights is build
5970             % when transforms with the key kashida= are declared.

```

```

5971     if Babel.kashida_wts then
5972         local k_wt = node.get_attribute(n, KASHIDA)
5973         if k_wt > 0 then % todo. parameter for multi inserts
5974             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5975         end
5976     end
5977
5978 end % of node.traverse_id
5979
5980 if #elongs == 0 and #k_list == 0 then goto next_line end
5981 full = line.width
5982 shift = line.shift
5983 goal = full * Babel.arabic.justify_factor % A bit crude
5984 width = node.dimensions(line.head) % The 'natural' width
5985
5986 % == Elongated ==
5987 % Original idea taken from 'chickenize'
5988 while (#elongs > 0 and width < goal) do
5989     subst_done = true
5990     local x = #elongs
5991     local curr = elongs[x].node
5992     local oldchar = curr.char
5993     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5994     width = node.dimensions(line.head) % Check if the line is too wide
5995     % Substitute back if the line would be too wide and break:
5996     if width > goal then
5997         curr.char = oldchar
5998         break
5999     end
6000     % If continue, pop the just substituted node from the list:
6001     table.remove(elongs, x)
6002 end
6003
6004 % == Tatwil ==
6005 % Traverse the kashida node list so many times as required, until
6006 % the line is filled. The first pass adds a tatweel after each
6007 % node with kashida in the line, the second pass adds another one,
6008 % and so on. In each pass, add first the kashida with the highest
6009 % weight, then with lower weight and so on.
6010 if #k_list == 0 then goto next_line end
6011
6012 width = node.dimensions(line.head) % The 'natural' width
6013 k_curr = #k_list % Traverse backwards, from the end
6014 wt_pos = 1
6015
6016 while width < goal do
6017     subst_done = true
6018     k_item = k_list[k_curr].node
6019     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6020         d = node.copy(k_item)
6021         d.char = 0x0640
6022         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6023         d.xoffset = 0
6024         line.head, new = node.insert_after(line.head, k_item, d)
6025         width_new = node.dimensions(line.head)
6026         if width > goal or width == width_new then
6027             node.remove(line.head, new) % Better compute before
6028             break
6029         end
6030         if Babel.fix_diacr then
6031             Babel.fix_diacr(k_item.next)
6032         end
6033         width = width_new

```

```

6034     end
6035     if k_curr == 1 then
6036         k_curr = #k_list
6037         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6038     else
6039         k_curr = k_curr - 1
6040     end
6041 end
6042
6043 % Limit the number of tatweel by removing them. Not very efficient,
6044 % but it does the job in a quite predictable way.
6045 if Babel.arabic.kashida_limit > -1 then
6046     cnt = 0
6047     for n in node.traverse_id(GLYPH, line.head) do
6048         if n.char == 0x0640 then
6049             cnt = cnt + 1
6050             if cnt > Babel.arabic.kashida_limit then
6051                 node.remove(line.head, n)
6052             end
6053         else
6054             cnt = 0
6055         end
6056     end
6057 end
6058
6059 ::next_line::
6060
6061 % Must take into account marks and ins, see luatex manual.
6062 % Have to be executed only if there are changes. Investigate
6063 % what's going on exactly.
6064 if subst_done and not gc then
6065     d = node.hpack(line.head, full, 'exactly')
6066     d.shift = shift
6067     node.insert_before(head, line, d)
6068     node.remove(head, line)
6069 end
6070 end % if process line
6071 end
6072 }
6073 \endgroup
6074 \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`.

```

6075 \def\bbl@scr@node@list{%
6076   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6077   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6078 \ifnum\bbl@bidimode=102 % bidi-r
6079   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6080 \fi
6081 \def\bbl@set@renderer{%
6082   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6083   \ifin@
6084     \let\bbl@unset@renderer\relax
6085   \else
6086     \bbl@exp{%
6087       \def\\bbl@unset@renderer{%
6088         \def<g__fontspec_default_fontopts_clist>{%

```



```

6089         \[g__fontspec_default_fontopts_clist]]}%
6090     \def\<g__fontspec_default_fontopts_clist>{%
6091         Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]]}%
6092     \fi}
6093 <@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.

```

6094 \directlua{% DL6
6095 Babel.script_blocks = {
6096   ['dflt'] = {},
6097   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6098               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6099   ['Armn'] = {{0x0530, 0x058F}},
6100   ['Beng'] = {{0x0980, 0x09FF}},
6101   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6102   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6103   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6104              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6105   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6106   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6107              {0xAB00, 0xAB2F}},
6108   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6109   % Don't follow strictly Unicode, which places some Coptic letters in
6110   % the 'Greek and Coptic' block
6111   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6112   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6113              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6114              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6115              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6116              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6117              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6118   ['Hebr'] = {{0x0590, 0x05FF},
6119              {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6120   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6121              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6122   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6123   ['Knda'] = {{0x0C80, 0x0CFF}},
6124   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6125              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6126              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6127   ['Lao'] = {{0x0E80, 0x0EFF}},
6128   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6129              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6130              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6131   ['Mahj'] = {{0x11150, 0x1117F}},
6132   ['Mlym'] = {{0x0D00, 0x0D7F}},
6133   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6134   ['Orya'] = {{0x0B00, 0x0B7F}},
6135   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6136   ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6137   ['Taml'] = {{0x0B80, 0x0BFF}},
6138   ['Telu'] = {{0x0C00, 0x0C7F}},
6139   ['Tfng'] = {{0x2D30, 0x2D7F}},
6140   ['Thai'] = {{0x0E00, 0x0EFF}},

```

```

6141 ['Tibt'] = {{0x0F00, 0x0FFF}},
6142 ['Vaii'] = {{0xA500, 0xA63F}},
6143 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6144 }
6145
6146 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6147 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6148 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6149
6150 function Babel.locale_map(head)
6151   if not Babel.locale_mapped then return head end
6152
6153   local LOCALE = Babel.attr_locale
6154   local GLYPH = node.id('glyph')
6155   local inmath = false
6156   local toloc_save
6157   for item in node.traverse(head) do
6158     local toloc
6159     if not inmath and item.id == GLYPH then
6160       % Optimization: build a table with the chars found
6161       if Babel.chr_to_loc[item.char] then
6162         toloc = Babel.chr_to_loc[item.char]
6163       else
6164         for lc, maps in pairs(Babel.loc_to_scr) do
6165           for _, rg in pairs(maps) do
6166             if item.char >= rg[1] and item.char <= rg[2] then
6167               Babel.chr_to_loc[item.char] = lc
6168               toloc = lc
6169               break
6170             end
6171           end
6172         end
6173         % Treat composite chars in a different fashion, because they
6174         % 'inherit' the previous locale.
6175         if (item.char >= 0x0300 and item.char <= 0x036F) or
6176            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6177            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6178           Babel.chr_to_loc[item.char] = -2000
6179           toloc = -2000
6180         end
6181         if not toloc then
6182           Babel.chr_to_loc[item.char] = -1000
6183         end
6184       end
6185       if toloc == -2000 then
6186         toloc = toloc_save
6187       elseif toloc == -1000 then
6188         toloc = nil
6189       end
6190       if toloc and Babel.locale_props[toloc] and
6191          Babel.locale_props[toloc].letters and
6192          tex.getcatcode(item.char) \string~= 11 then
6193         toloc = nil
6194       end
6195       if toloc and Babel.locale_props[toloc].script
6196          and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6197          and Babel.locale_props[toloc].script ==
6198          Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6199         toloc = nil
6200       end
6201       if toloc then
6202         if Babel.locale_props[toloc].lg then
6203           item.lang = Babel.locale_props[toloc].lg

```

```

6204         node.set_attribute(item, LOCALE, toloc)
6205     end
6206     if Babel.locale_props[toloc]['/'..item.font] then
6207         item.font = Babel.locale_props[toloc]['/'..item.font]
6208     end
6209 end
6210 toloc_save = toloc
6211 elseif not inmath and item.id == 7 then % Apply recursively
6212     item.replace = item.replace and Babel.locale_map(item.replace)
6213     item.pre      = item.pre and Babel.locale_map(item.pre)
6214     item.post     = item.post and Babel.locale_map(item.post)
6215 elseif item.id == node.id'math' then
6216     inmath = (item.subtype == 0)
6217 end
6218 end
6219 return head
6220 end
6221 }

```

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

```

6222 \newcommand\babelcharproperty[1]{%
6223   \count@=#1\relax
6224   \ifvmode
6225     \expandafter\bbl@chprop
6226   \else
6227     \bbl@error{charproperty-only-vertical}{}{}{}%
6228   \fi}
6229 \newcommand\bbl@chprop[3][\the\count@]{%
6230   \@tempcnta=#1\relax
6231   \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6232   {\bbl@error{unknown-char-property}{}{#2}{}%
6233   }%
6234   \loop
6235     \bbl@cs{chprop@#2}{#3}%
6236   \ifnum\count@<\@tempcnta
6237     \advance\count@\@ne
6238   \repeat}
6239 %
6240 \def\bbl@chprop@direction#1{%
6241   \directlua{
6242     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6243     Babel.characters[\the\count@]['d'] = '#1'
6244   }}
6245 \let\bbl@chprop@bc\bbl@chprop@direction
6246 %
6247 \def\bbl@chprop@mirror#1{%
6248   \directlua{
6249     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6250     Babel.characters[\the\count@]['m'] = '\number#1'
6251   }}
6252 \let\bbl@chprop@bmg\bbl@chprop@mirror
6253 %
6254 \def\bbl@chprop@linebreak#1{%
6255   \directlua{
6256     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6257     Babel.cjk_characters[\the\count@]['c'] = '#1'
6258   }}
6259 \let\bbl@chprop@lb\bbl@chprop@linebreak
6260 %
6261 \def\bbl@chprop@locale#1{%
6262   \directlua{
6263     Babel.chr_to_loc = Babel.chr_to_loc or {}

```

```

6264   Babel.chr_to_loc[\the\count@] =
6265   \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6266 }

```

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.

```

6267 \directlua{% DL7
6268   Babel.nohyphenation = \the\l@nohyphenation
6269 }

```

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

```

6270 \begingroup
6271 \catcode\~ =12
6272 \catcode\% =12
6273 \catcode\& =14
6274 \catcode\_| =12
6275 \gdef\babelprehyphenation{%&
6276   \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6277 \gdef\babelposthyphenation{%&
6278   \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6279 %
6280 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6281   \ifcase#1
6282     \bbl@activateprehyphen
6283   \or
6284     \bbl@activateposthyphen
6285   \fi
6286 \begingroup
6287   \def\babeltempa{\bbl@add@list\babeltempb}%&
6288   \let\babeltempb\empty
6289   \def\bbl@tempa{#5}%&
6290   \bbl@replace\bbl@tempa{,}{,}%& TODO. Ugly trick to preserve {}
6291   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
6292     \bbl@ifsamestring{##1}{remove}%&
6293     {\bbl@add@list\babeltempb{nil}}}%&
6294   {\directlua{
6295     local rep = [= [#1]=]
6296     local three_args = '%s*=%s*([%-d%.%a{ }|]+)%s+([%-d%.%a{ }|]+)%s+([%-d%.%a{ }|]+)'
6297     &% Numeric passes directly: kern, penalty...
6298     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6299     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6300     rep = rep:gsub('^%s*(after)%s*', 'after = true, ')
6301     rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
6302     rep = rep:gsub('node%s*=%s*([a+])%s*([a+])', Babel.capture_node)
6303     rep = rep:gsub(' (norule)' .. three_args,
6304       'norule = {' .. '%2, %3, %4' .. '}')
6305     if #1 == 0 or #1 == 2 then
6306       rep = rep:gsub(' (space)' .. three_args,
6307         'space = {' .. '%2, %3, %4' .. '}')
6308       rep = rep:gsub(' (spacefactor)' .. three_args,
6309         'spacefactor = {' .. '%2, %3, %4' .. '}')
6310       rep = rep:gsub(' (kashida)%s*=%s*([%s,]*)', Babel.capture_kashida)
6311       &% Transform values
6312       rep, n = rep:gsub(' {([%a-%.]+)|([%a_%.]+)}',
6313         function(v,d)
6314           return string.format (

```

```

6315         '\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6316         v,
6317         load( 'return Babel.locale_props'..
6318             '\the\csname bbl@id@@#3\endcsname].' .. d() )
6319     end )
6320     rep, n = rep:gsub( '{{(%a%-%.|+)|([%-d%.|+])}}',
6321         '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6322 end
6323 if #1 == 1 then
6324     rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6325     rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6326     rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6327 end
6328 tex.print([[\\string\babeltempa{[] .. rep .. [{}]])
6329 }}&%
6330 \bbl@foreach\babeltempb{&%
6331     \bbl@forkv{##1}{&%
6332         \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6333             post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6334         \ifin@else
6335             \bbl@error{bad-transform-option}{###1}{}&%
6336         \fi}&%
6337 \let\bbl@kv@attribute\relax
6338 \let\bbl@kv@label\relax
6339 \let\bbl@kv@fonts\empty
6340 \let\bbl@kv@prepend\relax
6341 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6342 \ifx\bbl@kv@fonts\empty\else\bbl@settransfont\fi
6343 \ifx\bbl@kv@attribute\relax
6344     \ifx\bbl@kv@label\relax\else
6345         \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6346         \bbl@replace\bbl@kv@fonts{ }{,}&%
6347         \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6348         \count@ \z@
6349         \def\bbl@elt##1##2##3{&%
6350             \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6351             {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6352                 {\count@ \@ne}&%
6353                 {\bbl@error{font-conflict-transforms}{}}}&%
6354             }}&%
6355         \bbl@transfont@list
6356         \ifnum\count@=\z@
6357             \bbl@exp{\global\bbl@add\bbl@transfont@list
6358                 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6359         \fi
6360         \bbl@ifunset{\bbl@kv@attribute}&%
6361         {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6362         {}&%
6363         \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6364     \fi
6365 \else
6366     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6367 \fi
6368 \directlua{
6369     local lbkr = Babel.linebreaking.replacements[#1]
6370     local u = unicode.utf8
6371     local id, attr, label
6372     if #1 == 0 then
6373         id = \the\csname bbl@id@@#3\endcsname\space
6374     else
6375         id = \the\csname l@#3\endcsname\space
6376     end
6377     \ifx\bbl@kv@attribute\relax

```

```

6378     attr = -1
6379 \else
6380     attr = luatexbase.registernumber'\bbl@kv@attribute'
6381 \fi
6382 \ifx\bbl@kv@label\relax\else &% Same refs:
6383     label = [==[\bbl@kv@label]==]
6384 \fi
6385 &% Convert pattern:
6386 local patt = string.gsub([==[#4]==], '%s', '')
6387 if #1 == 0 then
6388     patt = string.gsub(patt, '|', ' ')
6389 end
6390 if not u.find(patt, '()', nil, true) then
6391     patt = '()' .. patt .. '()'
6392 end
6393 if #1 == 1 then
6394     patt = string.gsub(patt, '%(%)%^', '^()')
6395     patt = string.gsub(patt, '%$(%)', '()$')
6396 end
6397 patt = u.gsub(patt, '{(.)}',
6398     function (n)
6399         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6400     end)
6401 patt = u.gsub(patt, '{(x%x%x%x+)}',
6402     function (n)
6403         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6404     end)
6405 lbkr[id] = lbkr[id] or {}
6406 table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6407     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6408 }&%
6409 \endgroup}
6410 \endgroup
6411 %
6412 \let\bbl@transfont@list\@empty
6413 \def\bbl@settransfont{%
6414     \global\let\bbl@settransfont\relax % Execute only once
6415 \gdef\bbl@transfont{%
6416     \def\bbl@elt####1####2####3{%
6417         \bbl@ifblank{####3}%
6418         {\count@ \tw@}% Do nothing if no fonts
6419         {\count@ \z@
6420         \bbl@vforeach{####3}{%
6421             \def\bbl@tempd{#####1}%
6422             \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6423             \ifx\bbl@tempd\bbl@tempe
6424                 \count@ \@ne
6425             \else\ifx\bbl@tempd\bbl@transfam
6426                 \count@ \@ne
6427             \fi\fi}%
6428         \ifcase\count@
6429             \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6430         \or
6431             \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6432         \fi}}%
6433     \bbl@transfont@list}%
6434 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6435 \gdef\bbl@transfam{-unknown-}%
6436 \bbl@foreach\bbl@font@fams{%
6437     \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6438     \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6439     {\xdef\bbl@transfam{##1}}%
6440     {}}}}

```

```

6441 %
6442 \DeclareRobustCommand\enablelocaletransform[1]{%
6443   \bbl@ifunset{bbl@ATR@#1@\language @}%
6444   {\bbl@error{transform-not-available}{#1}{}}}%
6445   {\bbl@csarg\setattribute{ATR@#1@\language @}{\@ne}}
6446 \DeclareRobustCommand\disablelocaletransform[1]{%
6447   \bbl@ifunset{bbl@ATR@#1@\language @}%
6448   {\bbl@error{transform-not-available-b}{#1}{}}}%
6449   {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}

```

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

```

6450 \def\bbl@activateposthyphen{%
6451   \let\bbl@activateposthyphen\relax
6452   \ifx\bbl@attr@hboxed\undefined
6453     \newattribute\bbl@attr@hboxed
6454   \fi
6455   \directlua{
6456     require('babel-transforms.lua')
6457     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6458   }}
6459 \def\bbl@activateprehyphen{%
6460   \let\bbl@activateprehyphen\relax
6461   \ifx\bbl@attr@hboxed\undefined
6462     \newattribute\bbl@attr@hboxed
6463   \fi
6464   \directlua{
6465     require('babel-transforms.lua')
6466     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6467   }}
6468 \newcommand\SetTransformValue[3]{%
6469   \directlua{
6470     Babel.locale_props[\the\csname bbl@id@#1\endcsname].vars["#2"] = #3
6471   }}

```

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

```

6472 \newcommand\ShowBabelTransforms[1]{%
6473   \bbl@activateprehyphen
6474   \bbl@activateposthyphen
6475   \begingroup
6476     \directlua{ Babel.show_transforms = true }%
6477     \setbox\z@\vbox{#1}%
6478     \directlua{ Babel.show_transforms = false }%
6479   \endgroup}

```

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

```

6480 \newcommand\localeprehyphenation[1]{%
6481   \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 `luaoftload` is applied, which is loaded by default by \TeX . Just in case, consider the possibility it has not been loaded.

```

6482 \def\bbl@activate@preotf{%
6483   \let\bbl@activate@preotf\relax % only once
6484   \directlua{

```

```

6485 function Babel.pre_otfload_v(head)
6486   if Babel.numbers and Babel.digits_mapped then
6487     head = Babel.numbers(head)
6488   end
6489   if Babel.bidi_enabled then
6490     head = Babel.bidi(head, false, dir)
6491   end
6492   return head
6493 end
6494 %
6495 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6496   if Babel.numbers and Babel.digits_mapped then
6497     head = Babel.numbers(head)
6498   end
6499   if Babel.bidi_enabled then
6500     head = Babel.bidi(head, false, dir)
6501   end
6502   return head
6503 end
6504 %
6505 luatexbase.add_to_callback('pre_linebreak_filter',
6506   Babel.pre_otfload_v,
6507   'Babel.pre_otfload_v',
6508   Babel.priority_in_callback('pre_linebreak_filter',
6509     'luaotfload.node_processor') or nil)
6510 %
6511 luatexbase.add_to_callback('hpack_filter',
6512   Babel.pre_otfload_h,
6513   'Babel.pre_otfload_h',
6514   Babel.priority_in_callback('hpack_filter',
6515     'luaotfload.node_processor') or nil)
6516 }}

```

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`.

```

6517 \breakafterdirmode=1
6518 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6519   \let\bbl@beforeforeign\leavevmode
6520   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6521   \RequirePackage{luatexbase}
6522   \bbl@activate@preotf
6523   \directlua{
6524     require('babel-data-bidi.lua')
6525     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6526       require('babel-bidi-basic.lua')
6527     \or
6528       require('babel-bidi-basic-r.lua')
6529     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6530     table.insert(Babel.ranges, {0xF0000, 0xFFFFD, 'on'})
6531     table.insert(Babel.ranges, {0x100000, 0x10FFFD, 'on'})
6532   \fi}
6533   \newattribute\bbl@attr@dir
6534   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6535   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6536 \fi
6537 %
6538 \chardef\bbl@thetextdir\z@
6539 \chardef\bbl@thepardir\z@
6540 \def\bbl@getluadir#1{%
6541   \directlua{
6542     if tex.#ldir == 'TLT' then

```



```

6543     tex.sprint('0')
6544 elseif tex.#ldir == 'TRT' then
6545     tex.sprint('1')
6546 else
6547     tex.sprint('0')
6548 end}}
6549 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6550 \ifcase#3\relax
6551 \ifcase\bbl@getluadir{#1}\relax\else
6552 #2 TLT\relax
6553 \fi
6554 \else
6555 \ifcase\bbl@getluadir{#1}\relax
6556 #2 TRT\relax
6557 \fi
6558 \fi}

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

6559 \def\bbl@thedir{0}
6560 \def\bbl@textdir#1{%
6561 \bbl@setluadir{text}\textdir{#1}%
6562 \chardef\bbl@thetextdir#1\relax
6563 \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6564 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6565 \def\bbl@pardir#1{% Used twice
6566 \bbl@setluadir{par}\pardir{#1}%
6567 \chardef\bbl@thepardir#1\relax}
6568 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6569 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6570 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

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

6571 \ifnum\bbl@bidimode>\z@ % Any bidi=
6572 \def\bbl@insidemath{0}%
6573 \def\bbl@everymath{\def\bbl@insidemath{1}}
6574 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6575 \frozen@everymath\expandafter{%
6576 \expandafter\bbl@everymath\the\frozen@everymath}
6577 \frozen@everydisplay\expandafter{%
6578 \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6579 \AtBeginDocument{
6580 \directlua{
6581 function Babel.math_box_dir(head)
6582 if not (token.get_macro('bbl@insidemath') == '0') then
6583 if Babel.hlist_has_bidi(head) then
6584 local d = node.new(node.id'dir')
6585 d.dir = '+TRT'
6586 node.insert_before(head, node.has_glyph(head), d)
6587 local inmath = false
6588 for item in node.traverse(head) do
6589 if item.id == 11 then
6590 inmath = (item.subtype == 0)
6591 elseif not inmath then
6592 node.set_attribute(item,
6593 Babel.attr_dir, token.get_macro('bbl@thedir'))
6594 end
6595 end
6596 end
6597 end
6598 return head
6599 end
6600 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,

```

```

6601      "Babel.math_box_dir", 0)
6602    if Babel.unset_atdir then
6603      luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6604        "Babel.unset_atdir")
6605      luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6606        "Babel.unset_atdir")
6607    end
6608  }}%
6609 \fi

Experimental. Tentative name.

6610 \DeclareRobustCommand\localebox[1]{%
6611   {\def\bbl@insidemath{0}%
6612    \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 `bidibasic`, 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.

```

6613 \bbl@trace{Redefinitions for bidi layout}
6614 %
6615 << *More package options >> ≡
6616 \chardef\bbl@eqnpos\z@
6617 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6618 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6619 << /More package options >>
6620 %
6621 \ifnum\bbl@bidimode>\z@ % Any bidi=
6622   \matheqdirmode\@ne % A luatex primitive
6623   \let\bbl@eqnodir\relax
6624   \def\bbl@eqdel{()}
6625   \def\bbl@eqnum{%
6626     {\normalfont\normalcolor
6627      \expandafter\@firstoftwo\bbl@eqdel
6628      \theequation
6629      \expandafter\@secondoftwo\bbl@eqdel}}
6630   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6631   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6632   \def\bbl@eqno@flip#1{%
6633     \ifdim\predisplaysize=-\maxdimen
6634       \eqno
6635       \hb@xt@.01pt{%
6636         \hb@xt@\displaywidth{\hss#1\glet\bbl@upset\@currentlabel}\hss}%
6637     \else
6638       \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6639   \fi

```

```

6640 \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6641 \def\bbl@leqno@flip#1{%
6642 \ifdim\predisplaysize=-\maxdimen
6643 \leqno
6644 \hb@xt@.01pt{%
6645 \hss\hb@xt@{\displaywidth{\#1\glet\bbl@upset\@currentlabel}\hss}}%
6646 \else
6647 \eqno\hbox{\#1\glet\bbl@upset\@currentlabel}%
6648 \fi
6649 \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6650 %
6651 \AtBeginDocument{%
6652 \ifx\bbl@noamsmath\relax\else
6653 \ifx\maketag@@@undefined % Normal equation, eqnarray
6654 \AddToHook{env/equation/begin}{%
6655 \ifnum\bbl@thetextdir>z@
6656 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6657 \let\@eqnnum\bbl@eqnum
6658 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6659 \chardef\bbl@thetextdirz@
6660 \bbl@add\normalfont{\bbl@eqnodir}%
6661 \ifcase\bbl@eqnpos
6662 \let\bbl@puteqno\bbl@eqno@flip
6663 \or
6664 \let\bbl@puteqno\bbl@leqno@flip
6665 \fi
6666 \fi}%
6667 \ifnum\bbl@eqnpos=\tw@ \else
6668 \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6669 \fi
6670 \AddToHook{env/eqnarray/begin}{%
6671 \ifnum\bbl@thetextdir>z@
6672 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6673 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6674 \chardef\bbl@thetextdirz@
6675 \bbl@add\normalfont{\bbl@eqnodir}%
6676 \ifnum\bbl@eqnpos=\@ne
6677 \def\@eqnnum{%
6678 \setbox\z@\hbox{\bbl@eqnum}%
6679 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6680 \else
6681 \let\@eqnnum\bbl@eqnum
6682 \fi
6683 \fi}
6684 % Hack for wrong vertical spacing with \[ \]. YA luatex bug?:
6685 \expandafter\bbl@sreplace\csname\endcsname{$$}{\eqno\kern.001pt$}$%
6686 \else % amstex
6687 \bbl@exp{% Hack to hide maybe undefined conditionals:
6688 \chardef\bbl@eqnpos=0%
6689 \<iftagsleft@>1<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6690 \ifnum\bbl@eqnpos=\@ne
6691 \let\bbl@ams@lap\hbox
6692 \else
6693 \let\bbl@ams@lap\llap
6694 \fi
6695 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6696 \bbl@sreplace\intertext@{\normalbaselines}%
6697 {\normalbaselines
6698 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6699 \ExplSyntaxOff
6700 \def\bbl@ams@tagbox#1#2{\#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6701 \ifx\bbl@ams@lap\hbox % leqno
6702 \def\bbl@ams@flip#1{%

```

```

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

```

Declarations specific to lua, called by \babelprovide.

```

6760 \def\bbl@provide@extra#1{%
6761     % == onchar ==
6762     \ifx\bbl@KVP@onchar\@nnil\else
6763         \bbl@luahyphenate

```

```

6764 \bbl@exp{%
6765   \\\AddToHook{env/document/before}{\\select@language{#1}}}%
6766 \directlua{
6767   if Babel.locale_mapped == nil then
6768     Babel.locale_mapped = true
6769     Babel.linebreaking.add_before(Babel.locale_map, 1)
6770     Babel.loc_to_scr = {}
6771     Babel.chr_to_loc = Babel.chr_to_loc or {}
6772   end
6773   Babel.locale_props[\the\localeid].letters = false
6774 }%
6775 \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6776 \ifin@
6777   \directlua{
6778     Babel.locale_props[\the\localeid].letters = true
6779   }%
6780 \fi
6781 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6782 \ifin@
6783   \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
6784     \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6785   \fi
6786   \bbl@exp{\\bbl@add\\bbl@starthyphens
6787     {\\bbl@patterns@lua{\language}}}%
6788   \directlua{
6789     if Babel.script_blocks['\bbl@cl{sbc}'] then
6790       Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6791       Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language}\space
6792     end
6793   }%
6794 \fi
6795 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6796 \ifin@
6797   \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
6798   \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
6799   \directlua{
6800     if Babel.script_blocks['\bbl@cl{sbc}'] then
6801       Babel.loc_to_scr[\the\localeid] =
6802         Babel.script_blocks['\bbl@cl{sbc}']
6803     end}%
6804   \ifx\bbl@mapselect\undefined
6805     \AtBeginDocument{%
6806       \bbl@patchfont{\bbl@mapselect}}%
6807     {\selectfont}}%
6808     \def\bbl@mapselect{%
6809       \let\bbl@mapselect\relax
6810       \edef\bbl@prefontid{\fontid\font}}%
6811     \def\bbl@mapdir##1{%
6812       \begingroup
6813         \setbox\z@\hbox{% Force text mode
6814           \def\language{##1}%
6815           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6816           \bbl@switchfont
6817           \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6818             \directlua{
6819               Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6820                 ['/\bbl@prefontid'] = \fontid\font\space}%
6821             \fi}%
6822           \endgroup}%
6823     \fi
6824     \bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\language}}}%
6825   \fi
6826 \fi

```

```

6827 % == mapfont ==
6828 % For bidi texts, to switch the font based on direction. Deprecated
6829 \ifx\bbl@KVP@mapfont\@nnil\else
6830   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}}%
6831   {\bbl@error{unknown-mapfont}}{}{}{}%
6832   \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys{language}}}%
6833   \bbl@ifunset{\bbl@wdir@language}{\bbl@provide@dirs{language}}}%
6834   \ifx\bbl@mapselect\@undefined
6835     \AtBeginDocument{%
6836       \bbl@patchfont{\bbl@mapselect}}%
6837       {\selectfont}}%
6838     \def\bbl@mapselect{%
6839       \let\bbl@mapselect\relax
6840       \edef\bbl@prefontid{\fontid\font}}%
6841     \def\bbl@mapdir##1{%
6842       {\def\language{##1}%
6843       \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6844       \bbl@switchfont
6845       \directlua{Babel.fontmap
6846         [\the\csname bbl@wdir@##1\endcsname]%
6847         [\bbl@prefontid]=\fontid\font}}}%
6848     \fi
6849     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{language}}}%
6850   \fi
6851 % == Line breaking: CJK quotes ==
6852 \ifcase\bbl@engine\or
6853   \bbl@xin@{/c}{\bbl@cl{lnbrk}}%
6854   \ifin@
6855     \bbl@ifunset{\bbl@quote@language}{}%
6856     {\directlua{
6857       Babel.locale_props[\the\localeid].cjk_quotes = {}
6858       local cs = 'op'
6859       for c in string.utfvalues(
6860         [[\csname bbl@quote@language\endcsname]]) do
6861         if Babel.cjk_characters[c].c == 'qu' then
6862           Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6863         end
6864         cs = ( cs == 'op') and 'cl' or 'op'
6865       end
6866     }}%
6867   \fi
6868 \fi
6869 % == Counters: mapdigits ==
6870 % Native digits
6871 \ifx\bbl@KVP@mapdigits\@nnil\else
6872   \bbl@ifunset{\bbl@dgnat@language}{}%
6873   {\bbl@activate@preotf
6874     \directlua{
6875       Babel.digits_mapped = true
6876       Babel.digits = Babel.digits or {}
6877       Babel.digits[\the\localeid] =
6878         table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6879       if not Babel.numbers then
6880         function Babel.numbers(head)
6881           local LOCALE = Babel.attr_locale
6882           local GLYPH = node.id'glyph'
6883           local inmath = false
6884           for item in node.traverse(head) do
6885             if not inmath and item.id == GLYPH then
6886               local temp = node.get_attribute(item, LOCALE)
6887               if Babel.digits[temp] then
6888                 local chr = item.char
6889                 if chr > 47 and chr < 58 then

```

```

6890             item.char = Babel.digits[temp][chr-47]
6891         end
6892     end
6893     elseif item.id == node.id'math' then
6894         inmath = (item.subtype == 0)
6895     end
6896 end
6897 return head
6898 end
6899 end
6900 }}%
6901 \fi
6902 % == transforms ==
6903 \ifx\bbbl@KVP@transforms\@nnil\else
6904   \def\bbbl@elt##1##2##3{%
6905     \in@{$transforms.}{$##1}%
6906     \ifin@
6907       \def\bbbl@tempa{##1}%
6908       \bbbl@replace\bbbl@tempa{transforms.}{}%
6909       \bbbl@carg\bbbl@transforms{babel\bbbl@tempa}{##2}{##3}%
6910     \fi}%
6911   \bbbl@exp{%
6912     \\bbbl@ifblank{\bbbl@cl{dgnat}}}%
6913     {\let\\bbbl@tempa\relax}%
6914     {\def\\bbbl@tempa{%
6915       \\bbbl@elt{transforms.prehyphenation}%
6916       {digits.native.1.0}{([0-9])}%
6917       \\bbbl@elt{transforms.prehyphenation}%
6918       {digits.native.1.1}{string={1string|0123456789|string|\bbbl@cl{dgnat}}}}}%
6919   \ifx\bbbl@tempa\relax\else
6920     \toks@{\expandafter\expandafter\expandafter{%
6921       \csname bbl@inidata@\language\endcsname}%
6922       \bbbl@csarg\edef{inidata@\language}{%
6923         \unexpanded\expandafter{\bbbl@tempa}%
6924         \the\toks@}%
6925     \fi
6926     \csname bbl@inidata@\language\endcsname
6927     \bbbl@release@transforms\relax % \relax closes the last item.
6928   \fi}

  Start tabular here:

6929 \def\localerestoredirs{%
6930   \ifcase\bbbl@thetextdir
6931     \ifnum\textdirection=\z@\else\textdir TLT\fi
6932   \else
6933     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6934   \fi
6935   \ifcase\bbbl@thepardir
6936     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6937   \else
6938     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6939   \fi}
6940 %
6941 \IfBabelLayout{tabular}%
6942   {\chardef\bbbl@tabular@mode\tw}% All RTL
6943   {\IfBabelLayout{notabular}%
6944     {\chardef\bbbl@tabular@mode\z}%
6945     {\chardef\bbbl@tabular@mode\@ne}}% Mixed, with LTR cols
6946 %
6947 \ifnum\bbbl@bidimode>\@ne % Any lua bidi= except default=1
6948   % Redefine: vrules mess up dirs.
6949   \def\@arstrut{\relax\copy\@arstrutbox}%
6950   \ifcase\bbbl@tabular@mode\or % 1 = Mixed - default

```

```

6951 \let\bbl@parabefore\relax
6952 \AddToHook{para/before}{\bbl@parabefore}
6953 \AtBeginDocument{%
6954   \bbl@replace\@tabular{$}{$%
6955     \def\bbl@insidemath{0}%
6956     \def\bbl@parabefore{\localerestoredirs}}%
6957   \ifnum\bbl@tabular@mode=\@one
6958     \bbl@ifunset{\@tabclassz}{}%
6959     \bbl@exp{% Hide conditionals
6960       \\bbl@sreplace\\ \@tabclassz
6961       {\<ifcase>\\ \@chnum}%
6962       {\ \\localerestoredirs\<ifcase>\\ \@chnum}}}%
6963   \@ifpackageloaded{colortbl}%
6964     {\bbl@sreplace\@classz
6965       {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6966   {\@ifpackageloaded{array}%
6967     {\bbl@exp{% Hide conditionals
6968       \\bbl@sreplace\\ \@classz
6969       {\<ifcase>\\ \@chnum}%
6970       {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6971       \\bbl@sreplace\\ \@classz
6972       {\ \\do@row@strut\<fi>}{\ \\do@row@strut\<fi>\egroup}}}%
6973     {}}%
6974   \fi}%
6975 \or % 2 = All RTL - tabular
6976 \let\bbl@parabefore\relax
6977 \AddToHook{para/before}{\bbl@parabefore}%
6978 \AtBeginDocument{%
6979   \@ifpackageloaded{colortbl}%
6980     {\bbl@replace\@tabular{$}{$%
6981       \def\bbl@insidemath{0}%
6982       \def\bbl@parabefore{\localerestoredirs}}%
6983     \bbl@sreplace\@classz
6984     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6985     {}}%
6986 \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.

```

6987 \AtBeginDocument{%
6988   \@ifpackageloaded{multicol}%
6989     {\toks@{\expandafter{\multi@column@out}}%
6990     \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6991   }%
6992   \@ifpackageloaded{paracol}%
6993     {\edef\pcol@output{%
6994       \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6995     {}}%
6996 \fi

```

Finish here if there in no layout.

```

6997 \ifx\bbl@opt@layout\@nnil\endinput\fi

```

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

```

6998 \ifnum\bbl@bidimode>\z@ % Any bidi=
6999 \def\bbl@nextfake#1% non-local changes, use always inside a group!
7000 \bbl@exp{%
7001   \mathdir\the\bodydir
7002   #1% Once entered in math, set boxes to restore values

```



```

7003 \def\\bbl@insidemath{0}%
7004 \<ifmmode>%
7005 \everyvbox{%
7006 \the\everyvbox
7007 \bodydir\the\bodydir
7008 \mathdir\the\mathdir
7009 \everyhbox{\the\everyhbox}%
7010 \everyvbox{\the\everyvbox}}%
7011 \everyhbox{%
7012 \the\everyhbox
7013 \bodydir\the\bodydir
7014 \mathdir\the\mathdir
7015 \everyhbox{\the\everyhbox}%
7016 \everyvbox{\the\everyvbox}}%
7017 \<fi>}}%
7018 \IfBabelLayout{nopars}
7019 {}
7020 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7021 \IfBabelLayout{pars}
7022 {\def\@hangfrom#1{%
7023 \setbox\@tempboxa\hbox{#1}}%
7024 \hangindent\wd\@tempboxa
7025 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
7026 \shapemode\@ne
7027 \fi
7028 \noindent\box\@tempboxa}}
7029 {}
7030 \fi
7031 %
7032 \IfBabelLayout{tabular}
7033 {\let\bbl@OL@tabular\@tabular
7034 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
7035 \let\bbl@NL@tabular\@tabular
7036 \AtBeginDocument{%
7037 \ifx\bbl@NL@tabular\@tabular\else
7038 \bbl@exp{\in{\bbl@nextfake}{\@tabular}}}%
7039 \ifin\else
7040 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
7041 \fi
7042 \let\bbl@NL@tabular\@tabular
7043 \fi}}
7044 {}
7045 %
7046 \IfBabelLayout{lists}
7047 {\let\bbl@OL@list\list
7048 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
7049 \let\bbl@NL@list\list
7050 \def\bbl@listparshape#1#2#3{%
7051 \parshape #1 #2 #3 %
7052 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
7053 \shapemode\tw@
7054 \fi}}
7055 {}
7056 %
7057 \IfBabelLayout{graphics}
7058 {\let\bbl@pictresetdir\relax
7059 \def\bbl@pictsetdir#1{%
7060 \ifcase\bbl@thetextdir
7061 \let\bbl@pictresetdir\relax
7062 \else
7063 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7064 \or\textdir TLT
7065 \else\bodydir TLT \textdir TLT

```

```

7066     \fi
7067     % \(\text|par)dir required in pgf:
7068     \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7069     \fi}%
7070 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
7071 \directlua{
7072     Babel.get_picture_dir = true
7073     Babel.picture_has_bidi = 0
7074     %
7075     function Babel.picture_dir (head)
7076         if not Babel.get_picture_dir then return head end
7077         if Babel.hlist_has_bidi(head) then
7078             Babel.picture_has_bidi = 1
7079         end
7080         return head
7081     end
7082     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7083         "Babel.picture_dir")
7084 }%
7085 \AtBeginDocument{%
7086     \def\LS@rot{%
7087         \setbox\@outputbox\vbox{%
7088             \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
7089     \long\def\put(#1,#2)#3{%
7090         \@killglue
7091         % Try:
7092         \ifx\bbl@pictresetdir\relax
7093             \def\bbl@tempc{0}%
7094         \else
7095             \directlua{
7096                 Babel.get_picture_dir = true
7097                 Babel.picture_has_bidi = 0
7098             }%
7099             \setbox\z@\hb@xt@\z@{%
7100                 \@defaultunitsset\@tempdimc{#1}\unitlength
7101                 \kern\@tempdimc
7102                 #3\hss}%
7103             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7104         \fi
7105         % Do:
7106         \@defaultunitsset\@tempdimc{#2}\unitlength
7107         \raise\@tempdimc\hb@xt@\z@{%
7108             \@defaultunitsset\@tempdimc{#1}\unitlength
7109             \kern\@tempdimc
7110             {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
7111         \ignorespaces}%
7112     \MakeRobust\put}%
7113 \AtBeginDocument
7114 {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir@gobble}%
7115 \ifx\pgfpicture\undefined\else
7116     \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
7117     \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
7118     \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
7119 \fi
7120 \ifx\tikzpicture\undefined\else
7121     \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
7122     \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
7123     \bbl@sreplace\tikz{\beginpgfgroup}{\beginpgfgroup\bbl@pictsetdir\tw@}%
7124     \bbl@sreplace\tikzpicture{\beginpgfgroup}{\beginpgfgroup\bbl@pictsetdir\tw@}%
7125 \fi
7126 \ifx\tcolorbox\undefined\else
7127     \def\tcb@drawing@env@begin{%
7128         \csname tcb@before@tcb@split@state\endcsname

```

```

7129      \bbl@pictsetdir\tw@
7130      \begin{\kvtcb@graphenv}%
7131      \tcbbddraw
7132      \tcb@apply@graph@patches}%
7133      \def\tcb@drawing@env@end{%
7134      \end{\kvtcb@graphenv}%
7135      \bbl@pictresetdir
7136      \csname tcb@after@\tcb@split@state\endcsname}%
7137      \fi
7138  }}
7139  {}

```

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`.

```

7140 \IfBabelLayout{counters*}%
7141 {\bbl@add\bbl@opt@layout{.counters.}%
7142  \directlua{
7143    luatexbase.add_to_callback("process_output_buffer",
7144      Babel.discard_sublr , "Babel.discard_sublr") }%
7145  }{}
7146 \IfBabelLayout{counters}%
7147 {\let\bbl@0L@@textsuperscript\@textsuperscript
7148  \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7149  \let\bbl@latinarabic=\@arabic
7150  \let\bbl@0L@@arabic\@arabic
7151  \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
7152  \ifpackagewith{babel}{bidi=default}%
7153    {\let\bbl@asciroman=\@roman
7154     \let\bbl@0L@@roman\@roman
7155     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
7156     \let\bbl@asciiRoman=\@Roman
7157     \let\bbl@0L@@roman\@Roman
7158     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
7159     \let\bbl@0L@labelenumii\labelenumii
7160     \def\labelenumii{}\theenumii}%
7161     \let\bbl@0L@p@enumiii\p@enumiii
7162     \def\p@enumiii{\p@enumii}\theenumii{}\{}{}{}

```

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

```

7163 \IfBabelLayout{extras}%
7164 {\bbl@ncarg\let\bbl@0L@underline{underline }%
7165  \bbl@carg\bbl@sreplace{underline }%
7166   {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
7167  \bbl@carg\bbl@sreplace{underline }%
7168   {\m@th$}{\m@th$\egroup}%
7169  \let\bbl@0L@LaTeXe\LaTeXe
7170  \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7171    \if b\expandafter\car\@series\@nil\boldmath\fi
7172    \babelsublr{%
7173      \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
7174  {}
7175 \end{luatex}

```

10.13 Lua: transforms

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

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the `luatex`

manual), we must convert it to a utf8 position. With first, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With last we must take into account the capture position points to the next character. Here word_head points to the starting node of the text to be matched.

```

7176 {*transforms
7177 Babel.linebreaking.replacements = {}
7178 Babel.linebreaking.replacements[0] = {} -- pre
7179 Babel.linebreaking.replacements[1] = {} -- post
7180
7181 function Babel.tovalue(v)
7182   if type(v) == 'table' then
7183     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7184   else
7185     return v
7186   end
7187 end
7188
7189 Babel.attr_hboxed = luatexbase.registernumber'bbl@attr@hboxed'
7190
7191 function Babel.set_hboxed(head, gc)
7192   for item in node.traverse(head) do
7193     node.set_attribute(item, Babel.attr_hboxed, 1)
7194   end
7195   return head
7196 end
7197
7198 Babel.fetch_subtext = {}
7199
7200 Babel.ignore_pre_char = function(node)
7201   return (node.lang == Babel.nohyphenation)
7202 end
7203
7204 Babel.show_transforms = false
7205
7206 -- Merging both functions doesn't seem feasible, because there are too
7207 -- many differences.
7208 Babel.fetch_subtext[0] = function(head)
7209   local word_string = ''
7210   local word_nodes = {}
7211   local lang
7212   local item = head
7213   local inmath = false
7214
7215   while item do
7216
7217     if item.id == 11 then
7218       inmath = (item.subtype == 0)
7219     end
7220
7221     if inmath then
7222       -- pass
7223     elseif item.id == 29 then
7224       local locale = node.get_attribute(item, Babel.attr_locale)
7225
7226       if lang == locale or lang == nil then
7227         lang = lang or locale
7228         if Babel.ignore_pre_char(item) then
7229           word_string = word_string .. Babel.us_char
7230         else
7231           if node.has_attribute(item, Babel.attr_hboxed) then
7232             word_string = word_string .. Babel.us_char
7233           else
7234

```

```

7235         word_string = word_string .. unicode.utf8.char(item.char)
7236     end
7237 end
7238 word_nodes[#word_nodes+1] = item
7239 else
7240     break
7241 end
7242
7243 elseif item.id == 12 and item.subtype == 13 then
7244     if node.has_attribute(item, Babel.attr_hboxed) then
7245         word_string = word_string .. Babel.us_char
7246     else
7247         word_string = word_string .. ' '
7248     end
7249     word_nodes[#word_nodes+1] = item
7250
7251     -- Ignore leading unrecognized nodes, too.
7252 elseif word_string ~= '' then
7253     word_string = word_string .. Babel.us_char
7254     word_nodes[#word_nodes+1] = item -- Will be ignored
7255 end
7256
7257 item = item.next
7258 end
7259
7260 -- Here and above we remove some trailing chars but not the
7261 -- corresponding nodes. But they aren't accessed.
7262 if word_string:sub(-1) == ' ' then
7263     word_string = word_string:sub(1,-2)
7264 end
7265 if Babel.show_transforms then texio.write_nl(word_string) end
7266 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7267 return word_string, word_nodes, item, lang
7268 end
7269
7270 Babel.fetch_subtext[1] = function(head)
7271     local word_string = ''
7272     local word_nodes = {}
7273     local lang
7274     local item = head
7275     local inmath = false
7276
7277     while item do
7278
7279         if item.id == 11 then
7280             inmath = (item.subtype == 0)
7281         end
7282
7283         if inmath then
7284             -- pass
7285
7286         elseif item.id == 29 then
7287             if item.lang == lang or lang == nil then
7288                 lang = lang or item.lang
7289                 if node.has_attribute(item, Babel.attr_hboxed) then
7290                     word_string = word_string .. Babel.us_char
7291                 elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7292                     word_string = word_string .. Babel.us_char
7293                 else
7294                     word_string = word_string .. unicode.utf8.char(item.char)
7295                 end
7296                 word_nodes[#word_nodes+1] = item
7297             else

```

```

7298         break
7299     end
7300
7301     elseif item.id == 7 and item.subtype == 2 then
7302         if node.has_attribute(item, Babel.attr_hboxed) then
7303             word_string = word_string .. Babel.us_char
7304         else
7305             word_string = word_string .. '='
7306         end
7307         word_nodes[#word_nodes+1] = item
7308
7309     elseif item.id == 7 and item.subtype == 3 then
7310         if node.has_attribute(item, Babel.attr_hboxed) then
7311             word_string = word_string .. Babel.us_char
7312         else
7313             word_string = word_string .. '|'
7314         end
7315         word_nodes[#word_nodes+1] = item
7316
7317     -- (1) Go to next word if nothing was found, and (2) implicitly
7318     -- remove leading USs.
7319     elseif word_string == '' then
7320         -- pass
7321
7322     -- This is the responsible for splitting by words.
7323     elseif (item.id == 12 and item.subtype == 13) then
7324         break
7325
7326     else
7327         word_string = word_string .. Babel.us_char
7328         word_nodes[#word_nodes+1] = item -- Will be ignored
7329     end
7330
7331     item = item.next
7332 end
7333 if Babel.show_transforms then texio.write_nl(word_string) end
7334 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7335 return word_string, word_nodes, item, lang
7336 end
7337
7338 function Babel.pre_hyphenate_replace(head)
7339     Babel.hyphenate_replace(head, 0)
7340 end
7341
7342 function Babel.post_hyphenate_replace(head)
7343     Babel.hyphenate_replace(head, 1)
7344 end
7345
7346 Babel.us_char = string.char(31)
7347
7348 function Babel.hyphenate_replace(head, mode)
7349     local u = unicode.utf8
7350     local lbkr = Babel.linebreaking.replacements[mode]
7351     local tovalue = Babel.tovalue
7352
7353     local word_head = head
7354
7355     if Babel.show_transforms then
7356         texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7357     end
7358
7359     while true do -- for each subtext block
7360

```

```

7361     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7362
7363     if Babel.debug then
7364         print()
7365         print((mode == 0) and '@@@<' or '@@@>', w)
7366     end
7367
7368     if nw == nil and w == '' then break end
7369
7370     if not lang then goto next end
7371     if not lbkr[lang] then goto next end
7372
7373     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7374     -- loops are nested.
7375     for k=1, #lbkr[lang] do
7376         local p = lbkr[lang][k].pattern
7377         local r = lbkr[lang][k].replace
7378         local attr = lbkr[lang][k].attr or -1
7379
7380         if Babel.debug then
7381             print('*****', p, mode)
7382         end
7383
7384         -- This variable is set in some cases below to the first *byte*
7385         -- after the match, either as found by u.match (faster) or the
7386         -- computed position based on sc if w has changed.
7387         local last_match = 0
7388         local step = 0
7389
7390         -- For every match.
7391         while true do
7392             if Babel.debug then
7393                 print('====')
7394             end
7395             local new -- used when inserting and removing nodes
7396             local dummy_node -- used by after
7397
7398             local matches = { u.match(w, p, last_match) }
7399
7400             if #matches < 2 then break end
7401
7402             -- Get and remove empty captures (with ()'s, which return a
7403             -- number with the position), and keep actual captures
7404             -- (from (...)), if any, in matches.
7405             local first = table.remove(matches, 1)
7406             local last = table.remove(matches, #matches)
7407             -- Non re-fetched substrings may contain \31, which separates
7408             -- subsubstrings.
7409             if string.find(w:sub(first, last-1), Babel.us_char) then break end
7410
7411             local save_last = last -- with A()BC()D, points to D
7412
7413             -- Fix offsets, from bytes to unicode. Explained above.
7414             first = u.len(w:sub(1, first-1)) + 1
7415             last = u.len(w:sub(1, last-1)) -- now last points to C
7416
7417             -- This loop stores in a small table the nodes
7418             -- corresponding to the pattern. Used by 'data' to provide a
7419             -- predictable behavior with 'insert' (w_nodes is modified on
7420             -- the fly), and also access to 'remove'd nodes.
7421             local sc = first-1 -- Used below, too
7422             local data_nodes = {}
7423

```

```

7424     local enabled = true
7425     for q = 1, last-first+1 do
7426         data_nodes[q] = w_nodes[sc+q]
7427         if enabled
7428             and attr > -1
7429             and not node.has_attribute(data_nodes[q], attr)
7430         then
7431             enabled = false
7432         end
7433     end
7434
7435     -- This loop traverses the matched substring and takes the
7436     -- corresponding action stored in the replacement list.
7437     -- sc = the position in substr nodes / string
7438     -- rc = the replacement table index
7439     local rc = 0
7440
7441     ----- TODO. dummy_node?
7442     while rc < last-first+1 or dummy_node do -- for each replacement
7443         if Babel.debug then
7444             print('.....', rc + 1)
7445         end
7446         sc = sc + 1
7447         rc = rc + 1
7448
7449         if Babel.debug then
7450             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7451             local ss = ''
7452             for itt in node.traverse(head) do
7453                 if itt.id == 29 then
7454                     ss = ss .. unicode.utf8.char(itt.char)
7455                 else
7456                     ss = ss .. '{' .. itt.id .. '}'
7457                 end
7458             end
7459             print('*****', ss)
7460
7461         end
7462
7463         local crep = r[rc]
7464         local item = w_nodes[sc]
7465         local item_base = item
7466         local placeholder = Babel.us_char
7467         local d
7468
7469         if crep and crep.data then
7470             item_base = data_nodes[crep.data]
7471         end
7472
7473         if crep then
7474             step = crep.step or step
7475         end
7476
7477         if crep and crep.after then
7478             crep.insert = true
7479             if dummy_node then
7480                 item = dummy_node
7481             else -- TODO. if there is a node after?
7482                 d = node.copy(item_base)
7483                 head, item = node.insert_after(head, item, d)
7484                 dummy_node = item
7485             end
7486         end

```



```

7487
7488     if crep and not crep.after and dummy_node then
7489         node.remove(head, dummy_node)
7490         dummy_node = nil
7491     end
7492
7493     if not enabled then
7494         last_match = save_last
7495         goto next
7496
7497     elseif crep and next(crep) == nil then -- = {}
7498         if step == 0 then
7499             last_match = save_last    -- Optimization
7500         else
7501             last_match = utf8.offset(w, sc+step)
7502         end
7503         goto next
7504
7505     elseif crep == nil or crep.remove then
7506         node.remove(head, item)
7507         table.remove(w_nodes, sc)
7508         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7509         sc = sc - 1 -- Nothing has been inserted.
7510         last_match = utf8.offset(w, sc+1+step)
7511         goto next
7512
7513     elseif crep and crep.kashida then -- Experimental
7514         node.set_attribute(item,
7515             Babel.attr_kashida,
7516             crep.kashida)
7517         last_match = utf8.offset(w, sc+1+step)
7518         goto next
7519
7520     elseif crep and crep.string then
7521         local str = crep.string(matches)
7522         if str == '' then -- Gather with nil
7523             node.remove(head, item)
7524             table.remove(w_nodes, sc)
7525             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7526             sc = sc - 1 -- Nothing has been inserted.
7527         else
7528             local loop_first = true
7529             for s in string.utfvalues(str) do
7530                 d = node.copy(item_base)
7531                 d.char = s
7532                 if loop_first then
7533                     loop_first = false
7534                     head, new = node.insert_before(head, item, d)
7535                     if sc == 1 then
7536                         word_head = head
7537                     end
7538                     w_nodes[sc] = d
7539                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7540                 else
7541                     sc = sc + 1
7542                     head, new = node.insert_before(head, item, d)
7543                     table.insert(w_nodes, sc, new)
7544                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7545                 end
7546                 if Babel.debug then
7547                     print('.....', 'str')
7548                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7549                 end

```

```

7550         end -- for
7551         node.remove(head, item)
7552     end -- if ''
7553     last_match = utf8.offset(w, sc+1+step)
7554     goto next
7555
7556 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7557     d = node.new(7, 3) -- (disc, regular)
7558     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7559     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7560     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7561     d.attr = item_base.attr
7562     if crep.pre == nil then -- TeXbook p96
7563         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7564     else
7565         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7566     end
7567     placeholder = '|'
7568     head, new = node.insert_before(head, item, d)
7569
7570 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7571     -- ERROR
7572
7573 elseif crep and crep.penalty then
7574     d = node.new(14, 0) -- (penalty, userpenalty)
7575     d.attr = item_base.attr
7576     d.penalty = tovalue(crep.penalty)
7577     head, new = node.insert_before(head, item, d)
7578
7579 elseif crep and crep.space then
7580     -- 655360 = 10 pt = 10 * 65536 sp
7581     d = node.new(12, 13) -- (glue, spaceskip)
7582     local quad = font.getfont(item_base.font).size or 655360
7583     node.setglue(d, tovalue(crep.space[1]) * quad,
7584                   tovalue(crep.space[2]) * quad,
7585                   tovalue(crep.space[3]) * quad)
7586     if mode == 0 then
7587         placeholder = ' '
7588     end
7589     head, new = node.insert_before(head, item, d)
7590
7591 elseif crep and crep.norule then
7592     -- 655360 = 10 pt = 10 * 65536 sp
7593     d = node.new(2, 3) -- (rule, empty) = \no*rule
7594     local quad = font.getfont(item_base.font).size or 655360
7595     d.width = tovalue(crep.norule[1]) * quad
7596     d.height = tovalue(crep.norule[2]) * quad
7597     d.depth = tovalue(crep.norule[3]) * quad
7598     head, new = node.insert_before(head, item, d)
7599
7600 elseif crep and crep.spacefactor then
7601     d = node.new(12, 13) -- (glue, spaceskip)
7602     local base_font = font.getfont(item_base.font)
7603     node.setglue(d,
7604                 tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7605                 tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7606                 tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7607     if mode == 0 then
7608         placeholder = ' '
7609     end
7610     head, new = node.insert_before(head, item, d)
7611
7612 elseif mode == 0 and crep and crep.space then

```

```

7613         -- ERROR
7614
7615     elseif crep and crep.kern then
7616         d = node.new(13, 1)      -- (kern, user)
7617         local quad = font.getfont(item_base.font).size or 655360
7618         d.attr = item_base.attr
7619         d.kern = tovalue(crep.kern) * quad
7620         head, new = node.insert_before(head, item, d)
7621
7622     elseif crep and crep.node then
7623         d = node.new(crep.node[1], crep.node[2])
7624         d.attr = item_base.attr
7625         head, new = node.insert_before(head, item, d)
7626
7627     end -- i.e., replacement cases
7628
7629     -- Shared by disc, space(factor), kern, node and penalty.
7630     if sc == 1 then
7631         word_head = head
7632     end
7633     if crep.insert then
7634         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7635         table.insert(w_nodes, sc, new)
7636         last = last + 1
7637     else
7638         w_nodes[sc] = d
7639         node.remove(head, item)
7640         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7641     end
7642
7643     last_match = utf8.offset(w, sc+1+step)
7644
7645     ::next::
7646
7647     end -- for each replacement
7648
7649     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7650     if Babel.debug then
7651         print('.....', '/')
7652         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7653     end
7654
7655     if dummy_node then
7656         node.remove(head, dummy_node)
7657         dummy_node = nil
7658     end
7659
7660     end -- for match
7661
7662     end -- for patterns
7663
7664     ::next::
7665     word_head = nw
7666     end -- for substring
7667
7668     if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7669     return head
7670 end
7671
7672 -- This table stores capture maps, numbered consecutively
7673 Babel.capture_maps = {}
7674
7675 -- The following functions belong to the next macro

```

```

7676 function Babel.capture_func(key, cap)
7677   local ret = "[" .. cap:gsub('{{[0-9]}}', "")..m[1]..[""] .. "]"
7678   local cnt
7679   local u = unicode.utf8
7680   ret, cnt = ret:gsub('{{[0-9]}|([^\^+]|(.-))}', Babel.capture_func_map)
7681   if cnt == 0 then
7682     ret = u.gsub(ret, '{{(%%X%%X%%X+}}',
7683       function (n)
7684         return u.char(tonumber(n, 16))
7685       end)
7686   end
7687   ret = ret:gsub("%[%]%%.%.%", '')
7688   ret = ret:gsub("%.%.%[%]%%", '')
7689   return key .. "[=function(m) return ] .. ret .. [ end]]
7690 end
7691
7692 function Babel.capt_map(from, mapno)
7693   return Babel.capture_maps[mapno][from] or from
7694 end
7695
7696 -- Handle the {n|abc|ABC} syntax in captures
7697 function Babel.capture_func_map(capno, from, to)
7698   local u = unicode.utf8
7699   from = u.gsub(from, '{{(%%X%%X%%X+}}',
7700     function (n)
7701       return u.char(tonumber(n, 16))
7702     end)
7703   to = u.gsub(to, '{{(%%X%%X%%X+}}',
7704     function (n)
7705       return u.char(tonumber(n, 16))
7706     end)
7707   local froms = {}
7708   for s in string.utfcharacters(from) do
7709     table.insert(froms, s)
7710   end
7711   local cnt = 1
7712   table.insert(Babel.capture_maps, {})
7713   local mlen = table.getn(Babel.capture_maps)
7714   for s in string.utfcharacters(to) do
7715     Babel.capture_maps[mlen][froms[cnt]] = s
7716     cnt = cnt + 1
7717   end
7718   return "]"..Babel.capt_map(m[" .. capno .. "," ..
7719     (mlen) .. "]).." .. "[["
7720 end
7721
7722 -- Create/Extend reversed sorted list of kashida weights:
7723 function Babel.capture_kashida(key, wt)
7724   wt = tonumber(wt)
7725   if Babel.kashida_wts then
7726     for p, q in ipairs(Babel.kashida_wts) do
7727       if wt == q then
7728         break
7729       elseif wt > q then
7730         table.insert(Babel.kashida_wts, p, wt)
7731         break
7732       elseif table.getn(Babel.kashida_wts) == p then
7733         table.insert(Babel.kashida_wts, wt)
7734       end
7735     end
7736   else
7737     Babel.kashida_wts = { wt }
7738   end

```

```

7739 return 'kashida = ' .. wt
7740 end
7741
7742 function Babel.capture_node(id, subtype)
7743   local sbt = 0
7744   for k, v in pairs(node.subtypes(id)) do
7745     if v == subtype then sbt = k end
7746   end
7747   return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7748 end
7749
7750 -- Experimental: applies prehyphenation transforms to a string (letters
7751 -- and spaces).
7752 function Babel.string_prehyphenation(str, locale)
7753   local n, head, last, res
7754   head = node.new(8, 0) -- dummy (hack just to start)
7755   last = head
7756   for s in string.utfvalues(str) do
7757     if s == 20 then
7758       n = node.new(12, 0)
7759     else
7760       n = node.new(29, 0)
7761       n.char = s
7762     end
7763     node.set_attribute(n, Babel.attr_locale, locale)
7764     last.next = n
7765     last = n
7766   end
7767   head = Babel.hyphenate_replace(head, 0)
7768   res = ''
7769   for n in node.traverse(head) do
7770     if n.id == 12 then
7771       res = res .. ' '
7772     elseif n.id == 29 then
7773       res = res .. unicode.utf8.char(n.char)
7774     end
7775   end
7776   tex.print(res)
7777 end
7778 /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.

```

7779 (*basic-r
7780 Babel.bidi_enabled = true
7781
7782 require('babel-data-bidi.lua')
7783
7784 local characters = Babel.characters
7785 local ranges = Babel.ranges
7786
7787 local DIR = node.id("dir")
7788
7789 local function dir_mark(head, from, to, outer)
7790   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse
7791   local d = node.new(DIR)
7792   d.dir = '+' .. dir
7793   node.insert_before(head, from, d)
7794   d = node.new(DIR)
7795   d.dir = '-' .. dir
7796   node.insert_after(head, to, d)
7797 end
7798
7799 function Babel.bidi(head, ispar)
7800   local first_n, last_n      -- first and last char with nums
7801   local last_es             -- an auxiliary 'last' used with nums
7802   local first_d, last_d     -- first and last char in L/R block
7803   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):

```

7804   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7805   local strong_lr = (strong == 'l') and 'l' or 'r'
7806   local outer = strong
7807
7808   local new_dir = false
7809   local first_dir = false
7810   local inmath = false
7811
7812   local last_lr
7813
7814   local type_n = ''
7815
7816   for item in node.traverse(head) do
7817
7818     -- three cases: glyph, dir, otherwise
7819     if item.id == node.id'glyph'
7820       or (item.id == 7 and item.subtype == 2) then
7821

```

```

7822     local itemchar
7823     if item.id == 7 and item.subtype == 2 then
7824         itemchar = item.replace.char
7825     else
7826         itemchar = item.char
7827     end
7828     local chardata = characters[itemchar]
7829     dir = chardata and chardata.d or nil
7830     if not dir then
7831         for nn, et in ipairs(ranges) do
7832             if itemchar < et[1] then
7833                 break
7834             elseif itemchar <= et[2] then
7835                 dir = et[3]
7836                 break
7837             end
7838         end
7839     end
7840     dir = dir or 'l'
7841     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).

```

7842     if new_dir then
7843         attr_dir = 0
7844         for at in node.traverse(item.attr) do
7845             if at.number == Babel.attr_dir then
7846                 attr_dir = at.value & 0x3
7847             end
7848         end
7849         if attr_dir == 1 then
7850             strong = 'r'
7851         elseif attr_dir == 2 then
7852             strong = 'al'
7853         else
7854             strong = 'l'
7855         end
7856         strong_lr = (strong == 'l') and 'l' or 'r'
7857         outer = strong_lr
7858         new_dir = false
7859     end
7860
7861     if dir == 'nsm' then dir = strong end -- W1

```

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

```

7862     dir_real = dir -- We need dir_real to set strong below
7863     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:

```

7864     if strong == 'al' then
7865         if dir == 'en' then dir = 'an' end -- W2
7866         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7867         strong_lr = 'r' -- W3
7868     end

```

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

```

7869     elseif item.id == node.id'dir' and not inmath then
7870         new_dir = true
7871         dir = nil
7872     elseif item.id == node.id'math' then

```

```

7873     inmath = (item.subtype == 0)
7874   else
7875     dir = nil          -- Not a char
7876   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>.

```

7877   if dir == 'en' or dir == 'an' or dir == 'et' then
7878     if dir ~= 'et' then
7879       type_n = dir
7880     end
7881     first_n = first_n or item
7882     last_n = last_es or item
7883     last_es = nil
7884   elseif dir == 'es' and last_n then -- W3+W6
7885     last_es = item
7886   elseif dir == 'cs' then          -- it's right - do nothing
7887   elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7888     if strong_lr == 'r' and type_n ~= '' then
7889       dir_mark(head, first_n, last_n, 'r')
7890     elseif strong_lr == 'l' and first_d and type_n == 'an' then
7891       dir_mark(head, first_n, last_n, 'r')
7892       dir_mark(head, first_d, last_d, outer)
7893       first_d, last_d = nil, nil
7894     elseif strong_lr == 'l' and type_n ~= '' then
7895       last_d = last_n
7896     end
7897     type_n = ''
7898     first_n, last_n = nil, nil
7899   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:

```

7900   if dir == 'l' or dir == 'r' then
7901     if dir ~= outer then
7902       first_d = first_d or item
7903       last_d = item
7904     elseif first_d and dir ~= strong_lr then
7905       dir_mark(head, first_d, last_d, outer)
7906       first_d, last_d = nil, nil
7907     end
7908   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.

```

7909   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7910     item.char = characters[item.char] and
7911       characters[item.char].m or item.char
7912   elseif (dir or new_dir) and last_lr ~= item then
7913     local mir = outer .. strong_lr .. (dir or outer)
7914     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7915       for ch in node.traverse(node.next(last_lr)) do
7916         if ch == item then break end
7917         if ch.id == node.id'glyph' and characters[ch.char] then
7918           ch.char = characters[ch.char].m or ch.char
7919         end

```



```

7920         end
7921     end
7922 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).

```

7923     if dir == 'l' or dir == 'r' then
7924         last_lr = item
7925         strong = dir_real          -- Don't search back - best save now
7926         strong_lr = (strong == 'l') and 'l' or 'r'
7927     elseif new_dir then
7928         last_lr = nil
7929     end
7930 end

```

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

```

7931     if last_lr and outer == 'r' then
7932         for ch in node.traverse_id(node.id('glyph', node.next(last_lr)) do
7933             if characters[ch.char] then
7934                 ch.char = characters[ch.char].m or ch.char
7935             end
7936         end
7937     end
7938     if first_n then
7939         dir_mark(head, first_n, last_n, outer)
7940     end
7941     if first_d then
7942         dir_mark(head, first_d, last_d, outer)
7943     end

```

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

```

7944     return node.prev(head) or head
7945 end
7946 </basic-r>

```

And here the Lua code for bidi=basic:

```

7947 <*>basic<*>
7948 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7949
7950 Babel.fontmap = Babel.fontmap or {}
7951 Babel.fontmap[0] = {}          -- l
7952 Babel.fontmap[1] = {}          -- r
7953 Babel.fontmap[2] = {}          -- al/an
7954
7955 -- To cancel mirroring. Also OML, OMS, U?
7956 Babel.symbol_fonts = Babel.symbol_fonts or {}
7957 Babel.symbol_fonts[font.id('tenln')] = true
7958 Babel.symbol_fonts[font.id('tenlnw')] = true
7959 Babel.symbol_fonts[font.id('tencirc')] = true
7960 Babel.symbol_fonts[font.id('tencircw')] = true
7961
7962 Babel.bidi_enabled = true
7963 Babel.mirroring_enabled = true
7964
7965 require('babel-data-bidi.lua')
7966
7967 local characters = Babel.characters
7968 local ranges = Babel.ranges
7969
7970 local DIR = node.id('dir')
7971 local GLYPH = node.id('glyph')
7972
7973 local function insert_implicit(head, state, outer)

```

```

7974 local new_state = state
7975 if state.sim and state.eim and state.sim ~= state.eim then
7976     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
7977     local d = node.new(DIR)
7978     d.dir = '+' .. dir
7979     node.insert_before(head, state.sim, d)
7980     local d = node.new(DIR)
7981     d.dir = '-' .. dir
7982     node.insert_after(head, state.eim, d)
7983 end
7984 new_state.sim, new_state.eim = nil, nil
7985 return head, new_state
7986 end
7987
7988 local function insert_numeric(head, state)
7989     local new
7990     local new_state = state
7991     if state.san and state.ean and state.san ~= state.ean then
7992         local d = node.new(DIR)
7993         d.dir = '+TLT'
7994         _, new = node.insert_before(head, state.san, d)
7995         if state.san == state.sim then state.sim = new end
7996         local d = node.new(DIR)
7997         d.dir = '-TLT'
7998         _, new = node.insert_after(head, state.ean, d)
7999         if state.ean == state.eim then state.eim = new end
8000     end
8001     new_state.san, new_state.ean = nil, nil
8002     return head, new_state
8003 end
8004
8005 local function glyph_not_symbol_font(node)
8006     if node.id == GLYPH then
8007         return not Babel.symbol_fonts[node.font]
8008     else
8009         return false
8010     end
8011 end
8012
8013 -- TODO - \hbox with an explicit dir can lead to wrong results
8014 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
8015 -- was made to improve the situation, but the problem is the 3-dir
8016 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8017 -- well.
8018
8019 function Babel.bidi(head, ispar, hdir)
8020     local d -- d is used mainly for computations in a loop
8021     local prev_d = ''
8022     local new_d = false
8023
8024     local nodes = {}
8025     local outer_first = nil
8026     local inmath = false
8027
8028     local glue_d = nil
8029     local glue_i = nil
8030
8031     local has_en = false
8032     local first_et = nil
8033
8034     local has_hyperlink = false
8035
8036     local ATDIR = Babel.attr_dir

```

```

8037 local attr_d, temp
8038 local locale_d
8039
8040 local save_outer
8041 local locale_d = node.get_attribute(head, ATDIR)
8042 if locale_d then
8043     locale_d = locale_d & 0x3
8044     save_outer = (locale_d == 0 and 'l') or
8045                  (locale_d == 1 and 'r') or
8046                  (locale_d == 2 and 'al')
8047 elseif ispar then -- Or error? Shouldn't happen
8048     -- when the callback is called, we are just _after_ the box,
8049     -- and the textdir is that of the surrounding text
8050     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8051 else -- Empty box
8052     save_outer = ('TRT' == hdir) and 'r' or 'l'
8053 end
8054 local outer = save_outer
8055 local last = outer
8056 -- 'al' is only taken into account in the first, current loop
8057 if save_outer == 'al' then save_outer = 'r' end
8058
8059 local fontmap = Babel.fontmap
8060
8061 for item in node.traverse(head) do
8062
8063     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8064     locale_d = node.get_attribute(item, ATDIR)
8065     node.set_attribute(item, ATDIR, 0x80)
8066
8067     -- In what follows, #node is the last (previous) node, because the
8068     -- current one is not added until we start processing the neutrals.
8069     -- three cases: glyph, dir, otherwise
8070     if glyph_not_symbol_font(item)
8071         or (item.id == 7 and item.subtype == 2) then
8072
8073         if locale_d == 0x80 then goto nextnode end
8074
8075         local d_font = nil
8076         local item_r
8077         if item.id == 7 and item.subtype == 2 then
8078             item_r = item.replace -- automatic discs have just 1 glyph
8079         else
8080             item_r = item
8081         end
8082
8083         local chardata = characters[item_r.char]
8084         d = chardata and chardata.d or nil
8085         if not d or d == 'nsm' then
8086             for nn, et in ipairs(ranges) do
8087                 if item_r.char < et[1] then
8088                     break
8089                 elseif item_r.char <= et[2] then
8090                     if not d then d = et[3]
8091                     elseif d == 'nsm' then d_font = et[3]
8092                     end
8093                     break
8094                 end
8095             end
8096         end
8097         d = d or 'l'
8098
8099         -- A short 'pause' in bidi for mapfont

```

```

8100      -- %%%% TODO. move if fontmap here
8101      d_font = d_font or d
8102      d_font = (d_font == 'l' and 0) or
8103                (d_font == 'nsm' and 0) or
8104                (d_font == 'r' and 1) or
8105                (d_font == 'al' and 2) or
8106                (d_font == 'an' and 2) or nil
8107      if d_font and fontmap and fontmap[d_font][item_r.font] then
8108          item_r.font = fontmap[d_font][item_r.font]
8109      end
8110
8111      if new_d then
8112          table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8113          if inmath then
8114              attr_d = 0
8115          else
8116              attr_d = locale_d & 0x3
8117          end
8118          if attr_d == 1 then
8119              outer_first = 'r'
8120              last = 'r'
8121          elseif attr_d == 2 then
8122              outer_first = 'r'
8123              last = 'al'
8124          else
8125              outer_first = 'l'
8126              last = 'l'
8127          end
8128          outer = last
8129          has_en = false
8130          first_et = nil
8131          new_d = false
8132      end
8133
8134      if glue_d then
8135          if (d == 'l' and 'l' or 'r') ~= glue_d then
8136              table.insert(nodes, {glue_i, 'on', nil})
8137          end
8138          glue_d = nil
8139          glue_i = nil
8140      end
8141
8142      elseif item.id == DIR then
8143          d = nil
8144          new_d = true
8145
8146      elseif item.id == node.id'glue' and item.subtype == 13 then
8147          glue_d = d
8148          glue_i = item
8149          d = nil
8150
8151      elseif item.id == node.id'math' then
8152          inmath = (item.subtype == 0)
8153
8154      elseif item.id == 8 and item.subtype == 19 then
8155          has_hyperlink = true
8156
8157      else
8158          d = nil
8159      end
8160
8161      -- AL <= EN/ET/ES      -- W2 + W3 + W6
8162      if last == 'al' and d == 'en' then

```

```

8163     d = 'an'          -- W3
8164 elseif last == 'al' and (d == 'et' or d == 'es') then
8165     d = 'on'          -- W6
8166 end
8167
8168 -- EN + CS/ES + EN      -- W4
8169 if d == 'en' and #nodes >= 2 then
8170     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8171         and nodes[#nodes-1][2] == 'en' then
8172         nodes[#nodes][2] = 'en'
8173     end
8174 end
8175
8176 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
8177 if d == 'an' and #nodes >= 2 then
8178     if (nodes[#nodes][2] == 'cs')
8179         and nodes[#nodes-1][2] == 'an' then
8180         nodes[#nodes][2] = 'an'
8181     end
8182 end
8183
8184 -- ET/EN                  -- W5 + W7->l / W6->on
8185 if d == 'et' then
8186     first_et = first_et or (#nodes + 1)
8187 elseif d == 'en' then
8188     has_en = true
8189     first_et = first_et or (#nodes + 1)
8190 elseif first_et then      -- d may be nil here !
8191     if has_en then
8192         if last == 'l' then
8193             temp = 'l'    -- W7
8194         else
8195             temp = 'en'   -- W5
8196         end
8197     else
8198         temp = 'on'      -- W6
8199     end
8200     for e = first_et, #nodes do
8201         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8202     end
8203     first_et = nil
8204     has_en = false
8205 end
8206
8207 -- Force mathdir in math if ON (currently works as expected only
8208 -- with 'l')
8209
8210 if inmath and d == 'on' then
8211     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8212 end
8213
8214 if d then
8215     if d == 'al' then
8216         d = 'r'
8217         last = 'al'
8218     elseif d == 'l' or d == 'r' then
8219         last = d
8220     end
8221     prev_d = d
8222     table.insert(nodes, {item, d, outer_first})
8223 end
8224
8225 outer_first = nil

```

```

8226
8227     ::nextnode::
8228
8229 end -- for each node
8230
8231 -- TODO -- repeated here in case EN/ET is the last node. Find a
8232 -- better way of doing things:
8233 if first_et then      -- dir may be nil here !
8234     if has_en then
8235         if last == 'l' then
8236             temp = 'l'    -- W7
8237         else
8238             temp = 'en'    -- W5
8239         end
8240     else
8241         temp = 'on'        -- W6
8242     end
8243     for e = first_et, #nodes do
8244         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8245     end
8246 end
8247
8248 -- dummy node, to close things
8249 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8250
8251 ----- NEUTRAL -----
8252
8253 outer = save_outer
8254 last = outer
8255
8256 local first_on = nil
8257
8258 for q = 1, #nodes do
8259     local item
8260
8261     local outer_first = nodes[q][3]
8262     outer = outer_first or outer
8263     last = outer_first or last
8264
8265     local d = nodes[q][2]
8266     if d == 'an' or d == 'en' then d = 'r' end
8267     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8268
8269     if d == 'on' then
8270         first_on = first_on or q
8271     elseif first_on then
8272         if last == d then
8273             temp = d
8274         else
8275             temp = outer
8276         end
8277         for r = first_on, q - 1 do
8278             nodes[r][2] = temp
8279             item = nodes[r][1]    -- MIRRORING
8280             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8281                 and temp == 'r' and characters[item.char] then
8282                 local font_mode = ''
8283                 if item.font > 0 and font.fonts[item.font].properties then
8284                     font_mode = font.fonts[item.font].properties.mode
8285                 end
8286                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8287                     item.char = characters[item.char].m or item.char
8288                 end

```

```

8289         end
8290     end
8291     first_on = nil
8292 end
8293
8294     if d == 'r' or d == 'l' then last = d end
8295 end
8296
8297 ----- IMPLICIT, REORDER -----
8298
8299 outer = save_outer
8300 last = outer
8301
8302 local state = {}
8303 state.has_r = false
8304
8305 for q = 1, #nodes do
8306     local item = nodes[q][1]
8307
8308     outer = nodes[q][3] or outer
8309
8310     local d = nodes[q][2]
8311
8312     if d == 'nsm' then d = last end          -- W1
8313     if d == 'en' then d = 'an' end
8314     local isdir = (d == 'r' or d == 'l')
8315
8316     if outer == 'l' and d == 'an' then
8317         state.san = state.san or item
8318         state.ean = item
8319     elseif state.san then
8320         head, state = insert_numeric(head, state)
8321     end
8322
8323     if outer == 'l' then
8324         if d == 'an' or d == 'r' then      -- im -> implicit
8325             if d == 'r' then state.has_r = true end
8326             state.sim = state.sim or item
8327             state.eim = item
8328         elseif d == 'l' and state.sim and state.has_r then
8329             head, state = insert_implicit(head, state, outer)
8330         elseif d == 'l' then
8331             state.sim, state.eim, state.has_r = nil, nil, false
8332         end
8333     end
8334     else
8335         if d == 'an' or d == 'l' then
8336             if nodes[q][3] then -- nil except after an explicit dir
8337                 state.sim = item -- so we move sim 'inside' the group
8338             else
8339                 state.sim = state.sim or item
8340             end
8341             state.eim = item
8342         elseif d == 'r' and state.sim then
8343             head, state = insert_implicit(head, state, outer)
8344         elseif d == 'r' then
8345             state.sim, state.eim = nil, nil
8346         end
8347     end
8348
8349     if isdir then
8350         last = d          -- Don't search back - best save now
8351     elseif d == 'on' and state.san then

```

```

8352     state.san = state.san or item
8353     state.ean = item
8354 end
8355
8356 end
8357
8358 head = node.prev(head) or head
8359 % \end{macrocode}
8360 %
8361 % Now direction nodes has been distributed with relation to characters
8362 % and spaces, we need to take into account \TeX-specific elements in
8363 % the node list, to move them at an appropriate place. Firstly, with
8364 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8365 % that the latter are still discardable.
8366 %
8367 % \begin{macrocode}
8368 --- FIXES ---
8369 if has_hyperlink then
8370   local flag, linking = 0, 0
8371   for item in node.traverse(head) do
8372     if item.id == DIR then
8373       if item.dir == '+TRT' or item.dir == '+TLT' then
8374         flag = flag + 1
8375       elseif item.dir == '-TRT' or item.dir == '-TLT' then
8376         flag = flag - 1
8377       end
8378       elseif item.id == 8 and item.subtype == 19 then
8379         linking = flag
8380       elseif item.id == 8 and item.subtype == 20 then
8381         if linking > 0 then
8382           if item.prev.id == DIR and
8383             (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8384             d = node.new(DIR)
8385             d.dir = item.prev.dir
8386             node.remove(head, item.prev)
8387             node.insert_after(head, item, d)
8388           end
8389         end
8390         linking = 0
8391       end
8392     end
8393   end
8394
8395   for item in node.traverse_id(10, head) do
8396     local p = item
8397     local flag = false
8398     while p.prev and p.prev.id == 14 do
8399       flag = true
8400       p = p.prev
8401     end
8402     if flag then
8403       node.insert_before(head, p, node.copy(item))
8404       node.remove(head, item)
8405     end
8406   end
8407
8408   return head
8409 end
8410
8410 function Babel.unset_atdir(head)
8411   local ATDIR = Babel.attr_dir
8412   for item in node.traverse(head) do
8413     node.set_attribute(item, ATDIR, 0x80)
8414   end

```



```

8415 return head
8416 end
8417 </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.

```

8418 <{*nil>
8419 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8420 \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.

```

8421 \ifx\l@nil\undefined
8422 \newlanguage\l@nil
8423 \namedef{bbl@hyphendata@the\l@nil}{}}}% Remove warning
8424 \let\bbl@elt\relax
8425 \edef\bbl@languages{% Add it to the list of languages
8426 \bbl@languages\bbl@elt{nil}{the\l@nil}{}}
8427 \fi

```

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

```

8428 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

`\captionnil`
`\datenil`

```

8429 \let\captionnil\@empty
8430 \let\datenil\@empty

```

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

```

8431 \def\bbl@inidata@nil{%
8432 \bbl@elt{identification}{tag.ini}{und}%
8433 \bbl@elt{identification}{load.level}{0}%
8434 \bbl@elt{identification}{charset}{utf8}%
8435 \bbl@elt{identification}{version}{1.0}%
8436 \bbl@elt{identification}{date}{2022-05-16}%
8437 \bbl@elt{identification}{name.local}{nil}%
8438 \bbl@elt{identification}{name.english}{nil}%
8439 \bbl@elt{identification}{name.babel}{nil}%
8440 \bbl@elt{identification}{tag.bcp47}{und}%
8441 \bbl@elt{identification}{language.tag.bcp47}{und}%

```

```

8442 \bbl@elt{identification}{tag.opentype}{dflt}%
8443 \bbl@elt{identification}{script.name}{Latin}%
8444 \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8445 \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8446 \bbl@elt{identification}{level}{1}%
8447 \bbl@elt{identification}{encodings}{}%
8448 \bbl@elt{identification}{derivate}{no}}
8449 \@namedef{bbl@tbc@nil}{und}
8450 \@namedef{bbl@lbc@nil}{und}
8451 \@namedef{bbl@casing@nil}{und}
8452 \@namedef{bbl@lotf@nil}{dflt}
8453 \@namedef{bbl@elname@nil}{nil}
8454 \@namedef{bbl@lname@nil}{nil}
8455 \@namedef{bbl@esname@nil}{Latin}
8456 \@namedef{bbl@sname@nil}{Latin}
8457 \@namedef{bbl@sbc@nil}{Latn}
8458 \@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.

```

8459 \ldf@finish{nil}
8460 </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.

```

8461 <<*Compute Julian day>> ≡
8462 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
8463 \def\bbl@cs@gregleap#1{%
8464   (\bbl@fpmmod{#1}{4} == 0) &&
8465   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
8466 \def\bbl@cs@jd#1#2#3{% year, month, day
8467   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8468     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8469     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8470     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3} }
8471 <</Compute Julian day>>

```

13.1. Islamic

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

```

8472 <*ca-islamic>
8473 \ExplSyntaxOn
8474 <@Compute Julian day>
8475 % == islamic (default)
8476 % Not yet implemented
8477 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar.

```

8478 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8479   ((#3 + ceil(29.5 * (#2 - 1)) +
8480     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8481     1948439.5) - 1) }
8482 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8483 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8484 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8485 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8486 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8487 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%

```

```

8488 \edef\bbl@tempa{%
8489 \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8490 \edef#5{%
8491 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8492 \edef#6{\fp_eval:n{
8493 min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8494 \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).

```

8495 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8496 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8497 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8498 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8499 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8500 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8501 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8502 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8503 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8504 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8505 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8506 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8507 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8508 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8509 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8510 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8511 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8512 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8513 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8514 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8515 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8516 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8517 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8518 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8519 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8520 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8521 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8522 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8523 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8524 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8525 65401,65431,65460,65490,65520}
8526 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8527 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8528 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8529 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8530 \ifnum#2>2014 \ifnum#2<2038
8531 \bbl@afterfi\expandafter\@gobble
8532 \fi\fi
8533 {\bbl@error{year-out-range}{2014-2038}{}}}%
8534 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
8535 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8536 \count@\@ne
8537 \bbl@foreach\bbl@cs@umalqura@data{%
8538 \advance\count@\@ne
8539 \ifnum##1>\bbl@tempd\else
8540 \edef\bbl@tempe{\the\count@}%
8541 \edef\bbl@tempb{##1}%
8542 \fi}%
8543 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month-lunar
8544 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8545 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%

```

```

8546 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8547 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8548 \ExplSyntaxOff
8549 \bbl@add\bbl@precalendar{%
8550 \bbl@replace\bbl@ld@calendar{-civil}}}%
8551 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8552 \bbl@replace\bbl@ld@calendar{+}}}%
8553 \bbl@replace\bbl@ld@calendar{-}}}%
8554 /ca-islamic

```

13.2. Hebrew

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

```

8555 \*ca-hebrew
8556 \newcount\bbl@cntcommon
8557 \def\bbl@remainder#1#2#3{%
8558 #3=#1\relax
8559 \divide #3 by #2\relax
8560 \multiply #3 by -#2\relax
8561 \advance #3 by #1\relax}%
8562 \newif\ifbbl@divisible
8563 \def\bbl@checkifdivisible#1#2{%
8564 {\countdef\tmp=0
8565 \bbl@remainder{#1}{#2}{\tmp}%
8566 \ifnum \tmp=0
8567 \global\bbl@divisibletrue
8568 \else
8569 \global\bbl@divisiblefalse
8570 \fi}}
8571 \newif\ifbbl@gregleap
8572 \def\bbl@ifgregleap#1{%
8573 \bbl@checkifdivisible{#1}{4}%
8574 \ifbbl@divisible
8575 \bbl@checkifdivisible{#1}{100}%
8576 \ifbbl@divisible
8577 \bbl@checkifdivisible{#1}{400}%
8578 \ifbbl@divisible
8579 \bbl@gregleaptrue
8580 \else
8581 \bbl@gregleapfalse
8582 \fi
8583 \else
8584 \bbl@gregleaptrue
8585 \fi
8586 \else
8587 \bbl@gregleapfalse
8588 \fi
8589 \ifbbl@gregleap}
8590 \def\bbl@gregdayspriormonths#1#2#3{%
8591 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8592 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8593 \bbl@ifgregleap{#2}%
8594 \ifnum #1 > 2
8595 \advance #3 by 1
8596 \fi
8597 \fi
8598 \global\bbl@cntcommon=#3}%
8599 #3=\bbl@cntcommon}
8600 \def\bbl@gregdaysprioryears#1#2{%
8601 {\countdef\tmpc=4
8602 \countdef\tmpb=2

```

```

8603 \tmpb=#1\relax
8604 \advance \tmpb by -1
8605 \tmpc=\tmpb
8606 \multiply \tmpc by 365
8607 #2=\tmpc
8608 \tmpc=\tmpb
8609 \divide \tmpc by 4
8610 \advance #2 by \tmpc
8611 \tmpc=\tmpb
8612 \divide \tmpc by 100
8613 \advance #2 by -\tmpc
8614 \tmpc=\tmpb
8615 \divide \tmpc by 400
8616 \advance #2 by \tmpc
8617 \global\bbl@cntcommon=#2\relax}%
8618 #2=\bbl@cntcommon}
8619 \def\bbl@absfromgreg#1#2#3#4{%
8620 {\countdef\tmpd=0
8621 #4=#1\relax
8622 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8623 \advance #4 by \tmpd
8624 \bbl@gregdaysprioryears{#3}{\tmpd}%
8625 \advance #4 by \tmpd
8626 \global\bbl@cntcommon=#4\relax}%
8627 #4=\bbl@cntcommon}
8628 \newif\ifbbl@hebrleap
8629 \def\bbl@checkleaphebryear#1{%
8630 {\countdef\tmpa=0
8631 \countdef\tmpb=1
8632 \tmpa=#1\relax
8633 \multiply \tmpa by 7
8634 \advance \tmpa by 1
8635 \bbl@remainder{\tmpa}{19}{\tmpb}%
8636 \ifnum \tmpb < 7
8637 \global\bbl@hebrleaptrue
8638 \else
8639 \global\bbl@hebrleapfalse
8640 \fi}}
8641 \def\bbl@hebreleapsedmonths#1#2{%
8642 {\countdef\tmpa=0
8643 \countdef\tmpb=1
8644 \countdef\tmpc=2
8645 \tmpa=#1\relax
8646 \advance \tmpa by -1
8647 #2=\tmpa
8648 \divide #2 by 19
8649 \multiply #2 by 235
8650 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8651 \tmpc=\tmpb
8652 \multiply \tmpb by 12
8653 \advance #2 by \tmpb
8654 \multiply \tmpc by 7
8655 \advance \tmpc by 1
8656 \divide \tmpc by 19
8657 \advance #2 by \tmpc
8658 \global\bbl@cntcommon=#2}%
8659 #2=\bbl@cntcommon}
8660 \def\bbl@hebreleapseddays#1#2{%
8661 {\countdef\tmpa=0
8662 \countdef\tmpb=1
8663 \countdef\tmpc=2
8664 \bbl@hebreleapsedmonths{#1}{#2}%
8665 \tmpa=#2\relax

```

```

8666 \multiply \tmpa by 13753
8667 \advance \tmpa by 5604
8668 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8669 \divide \tmpa by 25920
8670 \multiply #2 by 29
8671 \advance #2 by 1
8672 \advance #2 by \tmpa
8673 \bbl@remainder{#2}{7}{\tmpa}%
8674 \ifnum \tmpc < 19440
8675     \ifnum \tmpc < 9924
8676     \else
8677         \ifnum \tmpa=2
8678             \bbl@checkleaphebrewyear{#1}% of a common year
8679             \ifbbl@hebrleap
8680             \else
8681                 \advance #2 by 1
8682             \fi
8683         \fi
8684     \fi
8685     \ifnum \tmpc < 16789
8686     \else
8687         \ifnum \tmpa=1
8688             \advance #1 by -1
8689             \bbl@checkleaphebrewyear{#1}% at the end of leap year
8690             \ifbbl@hebrleap
8691                 \advance #2 by 1
8692             \fi
8693         \fi
8694     \fi
8695 \else
8696     \advance #2 by 1
8697 \fi
8698 \bbl@remainder{#2}{7}{\tmpa}%
8699 \ifnum \tmpa=0
8700     \advance #2 by 1
8701 \else
8702     \ifnum \tmpa=3
8703         \advance #2 by 1
8704     \else
8705         \ifnum \tmpa=5
8706             \advance #2 by 1
8707         \fi
8708     \fi
8709 \fi
8710 \global\bbl@cntcommon=#2\relax}%
8711 #2=\bbl@cntcommon}
8712 \def\bbl@daysinhebrewyear#1#2{%
8713     {\countdef\tmpe=12
8714     \bbl@hebrelapseddays{#1}{\tmpe}%
8715     \advance #1 by 1
8716     \bbl@hebrelapseddays{#1}{#2}%
8717     \advance #2 by -\tmpe
8718     \global\bbl@cntcommon=#2}%
8719 #2=\bbl@cntcommon}
8720 \def\bbl@hebrdayspriormonths#1#2#3{%
8721     {\countdef\tmpf= 14
8722     #3=\ifcase #1
8723         0 \or
8724         0 \or
8725         30 \or
8726         59 \or
8727         89 \or
8728         118 \or

```

```

8729         148 \or
8730         148 \or
8731         177 \or
8732         207 \or
8733         236 \or
8734         266 \or
8735         295 \or
8736         325 \or
8737         400
8738     \fi
8739     \bbl@checkleaphebreyear{#2}%
8740     \ifbbl@hebrleap
8741         \ifnum #1 > 6
8742             \advance #3 by 30
8743         \fi
8744     \fi
8745     \bbl@daysinhebreyear{#2}{\tmpf}%
8746     \ifnum #1 > 3
8747         \ifnum \tmpf=353
8748             \advance #3 by -1
8749         \fi
8750         \ifnum \tmpf=383
8751             \advance #3 by -1
8752         \fi
8753     \fi
8754     \ifnum #1 > 2
8755         \ifnum \tmpf=355
8756             \advance #3 by 1
8757         \fi
8758         \ifnum \tmpf=385
8759             \advance #3 by 1
8760         \fi
8761     \fi
8762     \global\bbl@cntcommon=#3\relax}%
8763     #3=\bbl@cntcommon}
8764 \def\bbl@absfromhebr#1#2#3#4{%
8765     {#4=#1\relax
8766     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8767     \advance #4 by #1\relax
8768     \bbl@hebrrelapseddays{#3}{#1}%
8769     \advance #4 by #1\relax
8770     \advance #4 by -1373429
8771     \global\bbl@cntcommon=#4\relax}%
8772     #4=\bbl@cntcommon}
8773 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8774     {\countdef\tmpx= 17
8775     \countdef\tmpy= 18
8776     \countdef\tmpz= 19
8777     #6=#3\relax
8778     \global\advance #6 by 3761
8779     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8780     \tmpz=1 \tmpy=1
8781     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8782     \ifnum \tmpx > #4\relax
8783         \global\advance #6 by -1
8784         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8785     \fi
8786     \advance #4 by -\tmpx
8787     \advance #4 by 1
8788     #5=#4\relax
8789     \divide #5 by 30
8790     \loop
8791         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%

```

```

8792      \ifnum \tmpx < #4\relax
8793      \advance #5 by 1
8794      \tmpy=\tmpx
8795      \repeat
8796      \global\advance #5 by -1
8797      \global\advance #4 by -\tmpy}}
8798 \newcount\bbbl@hebrday \newcount\bbbl@hebrmonth \newcount\bbbl@hebryear
8799 \newcount\bbbl@gregday \newcount\bbbl@gregmonth \newcount\bbbl@gregyear
8800 \def\bbbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8801   \bbbl@gregday=#3\relax \bbbl@gregmonth=#2\relax \bbbl@gregyear=#1\relax
8802   \bbbl@hebrfromgreg
8803   {\bbbl@gregday}{\bbbl@gregmonth}{\bbbl@gregyear}%
8804   {\bbbl@hebrday}{\bbbl@hebrmonth}{\bbbl@hebryear}%
8805   \edef#4{\the\bbbl@hebryear}%
8806   \edef#5{\the\bbbl@hebrmonth}%
8807   \edef#6{\the\bbbl@hebrday}}
8808 </ca-hebrew>

```

13.3. Persian

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

```

8809 <*ca-persian>
8810 \ExplSyntaxOn
8811 <@Compute Julian day@>
8812 \def\bbbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8813   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8814 \def\bbbl@ca@persian#1-#2-#3\@#4#5#6{%
8815   \edef\bbbl@tempa{#1}% 20XX-03-\bbbl@tempe = 1 farvardin:
8816   \ifnum\bbbl@tempa>2012 \ifnum\bbbl@tempa<2051
8817     \bbbl@afterfi\expandafter\@gobble
8818     \fi\fi
8819     {\bbbl@error{year-out-range}{2013-2050}{}}}%
8820 \bbbl@xin@{\bbbl@tempa}{\bbbl@cs@firstjal@xx}%
8821 \ifin@def\bbbl@tempe{20}\else\def\bbbl@tempe{21}\fi
8822 \edef\bbbl@tempc{\fp_eval:n{\bbbl@cs@jd{\bbbl@tempa}{#2}{#3}+.5}}% current
8823 \edef\bbbl@tempb{\fp_eval:n{\bbbl@cs@jd{\bbbl@tempa}{03}{\bbbl@tempe}+.5}}% begin
8824 \ifnum\bbbl@tempc<\bbbl@tempb
8825   \edef\bbbl@tempa{\fp_eval:n{\bbbl@tempa-1}}% go back 1 year and redo
8826   \bbbl@xin@{\bbbl@tempa}{\bbbl@cs@firstjal@xx}%
8827   \ifin@def\bbbl@tempe{20}\else\def\bbbl@tempe{21}\fi
8828   \edef\bbbl@tempb{\fp_eval:n{\bbbl@cs@jd{\bbbl@tempa}{03}{\bbbl@tempe}+.5}}%
8829   \fi
8830 \edef#4{\fp_eval:n{\bbbl@tempa-621}}% set Jalali year
8831 \edef#6{\fp_eval:n{\bbbl@tempc-\bbbl@tempb+1}}% days from 1 farvardin
8832 \edef#5{\fp_eval:n{% set Jalali month
8833   (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8834 \edef#6{\fp_eval:n{% set Jalali day
8835   (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}%
8836 \ExplSyntaxOff
8837 </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.

```

8838 <*ca-coptic>
8839 \ExplSyntaxOn
8840 <@Compute Julian day@>

```



```

8841 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8842 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8843 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8844 \edef#4{\fp_eval:n{%
8845 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8846 \edef\bbl@tempc{\fp_eval:n{%
8847 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8848 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8849 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8850 \ExplSyntaxOff
8851 \</ca-coptic
8852 \<ca-ethiopic
8853 \ExplSyntaxOn
8854 <@Compute Julian day@>
8855 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8856 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8857 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8858 \edef#4{\fp_eval:n{%
8859 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8860 \edef\bbl@tempc{\fp_eval:n{%
8861 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8862 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8863 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8864 \ExplSyntaxOff
8865 \</ca-ethiopic

```

13.5. Buddhist

That's very simple.

```

8866 \<ca-buddhist
8867 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8868 \edef#4{\number\numexpr#1+543\relax}%
8869 \edef#5{#2}%
8870 \edef#6{#3}}
8871 \</ca-buddhist
8872 %
8873 % \subsection{Chinese}
8874 %
8875 % Brute force, with the Julian day of first day of each month. The
8876 % table has been computed with the help of \textsf{python-lunardate} by
8877 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8878 % is 2015-2044.
8879 %
8880 % \begin{macrocode}
8881 \<ca-chinese
8882 \ExplSyntaxOn
8883 <@Compute Julian day@>
8884 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8885 \edef\bbl@tempd{\fp_eval:n{%
8886 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8887 \count@ \z@
8888 \@tempcnta=2015
8889 \bbl@foreach\bbl@cs@chinese@data{%
8890 \ifnum##1>\bbl@tempd\else
8891 \advance\count@\@ne
8892 \ifnum\count@>12
8893 \count@\@ne
8894 \advance\@tempcnta\@ne\fi
8895 \bbl@xin@{,##1,},{,\bbl@cs@chinese@leap,}%
8896 \ifin@
8897 \advance\count@\m@ne
8898 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8899 \else

```

```

8900      \edef\bbl@tempe{\the\count@}%
8901      \fi
8902      \edef\bbl@tempb{##1}%
8903      \fi}%
8904      \edef#4{\the\@tempcnta}%
8905      \edef#5{\bbl@tempe}%
8906      \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8907 \def\bbl@cs@chinese@leap{%
8908 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8909 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8910 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8911 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8912 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8913 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8914 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8915 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8916 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8917 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8918 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8919 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8920 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8921 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8922 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8923 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8924 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8925 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8926 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8927 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8928 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8929 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8930 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8931 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8932 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8933 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8934 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8935 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8936 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8937 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8938 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8939 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8940 10896,10926,10956,10986,11015,11045,11074,11103}
8941 \ExplSyntaxOff
8942 </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`.

```

8943 <{*bplain | blplain}
8944 \catcode`\{=1 % left brace is begin-group character
8945 \catcode`\}=2 % right brace is end-group character
8946 \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).

```

8947 \openin 0 hyphen.cfg
8948 \ifeof0
8949 \else
8950 \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.

```

8951 \def\input #1 {%
8952 \let\input\input
8953 \a hyphen.cfg
8954 \let\input\input
8955 }
8956 \fi
8957 </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`.

```

8958 <bplain>\a plain.tex
8959 <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.

```

8960 <bplain>\def\fmtname{babel-plain}
8961 <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).

```

8962 <<{*Emulate LaTeX} ≡
8963 \def\@empty{}
8964 \def\loadlocalcfg#1{%
8965 \openin0#1.cfg
8966 \ifeof0
8967 \closein0
8968 \else
8969 \closein0
8970 {\immediate\writel6{*****}%
8971 \immediate\writel6{* Local config file #1.cfg used}%
8972 \immediate\writel6{*}%
8973 }
8974 \input #1.cfg\relax
8975 \fi
8976 \@endofldf}

```

14.3. General tools

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

```

8977 \long\def\@firstofone#1{#1}

```

```

8978 \long\def\@firstoftwo#1#2{#1}
8979 \long\def\@secondoftwo#1#2{#2}
8980 \def\@nnil{\@nil}
8981 \def\@gobbletwo#1#2{}
8982 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8983 \def\@star@or@long#1{%
8984   \@ifstar
8985   {\let\l@ngrel@x\relax#1}%
8986   {\let\l@ngrel@x\long#1}}
8987 \let\l@ngrel@x\relax
8988 \def\@car#1#2\@nil{#1}
8989 \def\@cdr#1#2\@nil{#2}
8990 \let\@typeset@protect\relax
8991 \let\protected@edef\edef
8992 \long\def\@gobble#1{}
8993 \edef\@backslashchar{\expandafter\@gobble\string\}
8994 \def\strip@prefix#1>{}
8995 \def\g@addto@macro#1#2{%
8996   \toks@\expandafter{#1#2}%
8997   \xdef#1{\the\toks@}}
8998 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8999 \def\@nameuse#1{\csname #1\endcsname}
9000 \def\@ifundefined#1{%
9001   \expandafter\ifx\csname#1\endcsname\relax
9002     \expandafter\@firstoftwo
9003   \else
9004     \expandafter\@secondoftwo
9005   \fi}
9006 \def\@expandtwoargs#1#2#3{%
9007   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9008 \def\zap@space#1 #2{%
9009   #1%
9010   \ifx#2\@empty\else\expandafter\zap@space\fi
9011   #2}
9012 \let\bbl@trace\@gobble
9013 \def\bbl@error#1{% Implicit #2#3#4
9014   \begingroup
9015     \catcode`\=0 \catcode`\==12 \catcode`\=12
9016     \catcode`\^^M=5 \catcode`\%=14
9017     \input errbabel.def
9018   \endgroup
9019   \bbl@error{#1}}
9020 \def\bbl@warning#1{%
9021   \begingroup
9022     \newlinechar=`^^J
9023     \def\{^^J(babel) }%
9024     \message{\{#1}%
9025   \endgroup}
9026 \let\bbl@infowarn\bbl@warning
9027 \def\bbl@info#1{%
9028   \begingroup
9029     \newlinechar=`^^J
9030     \def\{^^J}%
9031     \wlog{#1}%
9032   \endgroup}

```

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

```

9033 \ifx\@preamblecmds\undefined
9034   \def\@preamblecmds{}
9035 \fi
9036 \def\@onlypreamble#1{%
9037   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%

```

```

9038 \@preamblecmds\do#1}}
9039 \@onlypreamble\@onlypreamble

```

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

```

9040 \def\begindocument{%
9041 \@begindocumenthook
9042 \global\let\@begindocumenthook\@undefined
9043 \def\do##1{\global\let##1\@undefined}%
9044 \@preamblecmds
9045 \global\let\do\noexpand}

9046 \ifx\@begindocumenthook\@undefined
9047 \def\@begindocumenthook{}
9048 \fi
9049 \@onlypreamble\@begindocumenthook
9050 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

9051 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
9052 \@onlypreamble\AtEndOfPackage
9053 \def\@endofldf{}
9054 \@onlypreamble\@endofldf
9055 \let\bbl@afterlang\@empty
9056 \chardef\bbl@opt@hyphenmap\z@

```

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

```

9057 \catcode`\&=\z@
9058 \ifx&\if@files\@undefined
9059 \expandafter\let\csname if@files\expandafter\endcsname
9060 \csname iffalse\endcsname
9061 \fi
9062 \catcode`\&=4

```

Mimic \LaTeX 's commands to define control sequences.

```

9063 \def\newcommand{\@star@or@long\new@command}
9064 \def\new@command#1{%
9065 \@testopt{\@newcommand#1}0}
9066 \def\@newcommand#1[#2]{%
9067 \@ifnextchar [{\@xargdef#1[#2]}%
9068 {\@argdef#1[#2]}}
9069 \long\def\@argdef#1[#2]#3{%
9070 \@yargdef#1\@ne{#2}{#3}}
9071 \long\def\@xargdef#1[#2][#3]#4{%
9072 \expandafter\def\expandafter#1\expandafter{%
9073 \expandafter\@protected@testopt\expandafter #1%
9074 \csname\string#1\expandafter\endcsname{#3}}%
9075 \expandafter\@yargdef \csname\string#1\endcsname
9076 \tw@{#2}{#4}}
9077 \long\def\@yargdef#1#2#3{%
9078 \@tempcnta#3\relax
9079 \advance \@tempcnta \@ne
9080 \let\@hash@\relax
9081 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9082 \@tempcntb #2%
9083 \@whilenum \@tempcntb < \@tempcnta
9084 \do{%
9085 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
9086 \advance\@tempcntb \@ne}%
9087 \let\@hash@##%
9088 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9089 \def\providecommand{\@star@or@long\provide@command}

```

```

9090 \def\providecommand#1{%
9091   \begingroup
9092     \escapechar\m@ne\xdef\@gtempa{\string#1}%
9093   \endgroup
9094   \expandafter\@ifundefined\@gtempa
9095     {\def\reserved@a{\newcommand#1}%
9096      {\let\reserved@a\relax
9097       \def\reserved@a{\newcommand\reserved@a}%
9098      \reserved@a}%
9099 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9100 \def\declare@robustcommand#1{%
9101   \edef\reserved@a{\string#1}%
9102   \def\reserved@b{#1}%
9103   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9104   \edef#1{%
9105     \ifx\reserved@a\reserved@b
9106       \noexpand\x@protect
9107       \noexpand#1%
9108     \fi
9109     \noexpand\protect
9110     \expandafter\noexpand\csname
9111       \expandafter\@gobble\string#1 \endcsname
9112   }%
9113   \expandafter\newcommand\csname
9114     \expandafter\@gobble\string#1 \endcsname
9115 }
9116 \def\x@protect#1{%
9117   \ifx\protect\@typeset@protect\else
9118     \@x@protect#1%
9119   \fi
9120 }
9121 \catcode`\&=\z@ % Trick to hide conditionals
9122 \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`.

```

9123 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9124 \catcode`\&=4
9125 \ifx\in@\@undefined
9126   \def\in@#1#2{%
9127     \def\in@##1#1##2##3\in@{%
9128       \ifx\in@##2\in@false\else\in@true\fi}%
9129     \in@##2#1\in@\in@}
9130 \else
9131   \let\bbl@tempa\@empty
9132 \fi
9133 \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).

```

9134 \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.

```

9135 \def\@ifl@aded#1#2#3#4{}

```

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

```

9136 \ifx\@tempcnta\@undefined
9137   \csname newcount\endcsname\@tempcnta\relax
9138 \fi
9139 \ifx\@tempcntb\@undefined
9140   \csname newcount\endcsname\@tempcntb\relax
9141 \fi

```

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

```

9142 \ifx\bye\@undefined
9143   \advance\count10 by -2\relax
9144 \fi
9145 \ifx\@ifnextchar\@undefined
9146   \def\@ifnextchar#1#2#3{%
9147     \let\reserved@d=#1%
9148     \def\reserved@a{#2}\def\reserved@b{#3}%
9149     \futurelet\@let@token\@ifnch}
9150 \def\@ifnch{%
9151   \ifx\@let@token\@sptoken
9152     \let\reserved@c\@xifnch
9153   \else
9154     \ifx\@let@token\reserved@d
9155       \let\reserved@c\reserved@a
9156     \else
9157       \let\reserved@c\reserved@b
9158     \fi
9159   \fi
9160   \reserved@c}
9161 \def\:\let\@sptoken= \: % this makes \@sptoken a space token
9162 \def\:\@xifnch\expandafter\def\:\{\futurelet\@let@token\@ifnch}
9163 \fi
9164 \def\@testopt#1#2{%
9165   \@ifnextchar[#{#1}{#1[#2]}}
9166 \def\@protected@testopt#1{%
9167   \ifx\protect\@typeset@protect
9168     \expandafter\@testopt
9169   \else
9170     \@x@protect#1%
9171   \fi}
9172 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9173   #2\relax}\fi}
9174 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9175   \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9176 \def\DeclareTextCommand{%
9177   \@dec@text@cmd\providecommand
9178 }
9179 \def\ProvideTextCommand{%
9180   \@dec@text@cmd\providecommand
9181 }
9182 \def\DeclareTextSymbol#1#2#3{%
9183   \@dec@text@cmd\chardef#1{#2}#3\relax
9184 }
9185 \def\@dec@text@cmd#1#2#3{%
9186   \expandafter\def\expandafter#2%
9187     \expandafter{%
9188       \csname#3-cmd\expandafter\endcsname
9189       \expandafter#2%
9190       \csname#3\string#2\endcsname
9191     }%

```

```

9192 % \let\@ifdefinable\@rc@ifdefinable
9193 \expandafter#1\csname#3\string#2\endcsname
9194 }
9195 \def\@current@cmd#1{%
9196 \ifx\protect\@typeset@protect\else
9197 \noexpand#1\expandafter\@gobble
9198 \fi
9199 }
9200 \def\@changed@cmd#1#2{%
9201 \ifx\protect\@typeset@protect
9202 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9203 \expandafter\ifx\csname ?\string#1\endcsname\relax
9204 \expandafter\def\csname ?\string#1\endcsname{%
9205 \@changed@x@err{#1}%
9206 }%
9207 \fi
9208 \global\expandafter\let
9209 \csname\cf@encoding\string#1\expandafter\endcsname
9210 \csname ?\string#1\endcsname
9211 \fi
9212 \csname\cf@encoding\string#1%
9213 \expandafter\endcsname
9214 \else
9215 \noexpand#1%
9216 \fi
9217 }
9218 \def\@changed@x@err#1{%
9219 \errhelp{Your command will be ignored, type <return> to proceed}%
9220 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9221 \def\DeclareTextCommandDefault#1{%
9222 \DeclareTextCommand#1?%
9223 }
9224 \def\ProvideTextCommandDefault#1{%
9225 \ProvideTextCommand#1?%
9226 }
9227 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9228 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9229 \def\DeclareTextAccent#1#2#3{%
9230 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
9231 }
9232 \def\DeclareTextCompositeCommand#1#2#3#4{%
9233 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9234 \edef\reserved@b{\string##1}%
9235 \edef\reserved@c{%
9236 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9237 \ifx\reserved@b\reserved@c
9238 \expandafter\expandafter\expandafter\ifx
9239 \expandafter\@car\reserved@a\relax\relax\@nil
9240 \@text@composite
9241 \else
9242 \edef\reserved@b##1{%
9243 \def\expandafter\noexpand
9244 \csname#2\string#1\endcsname###1{%
9245 \noexpand\@text@composite
9246 \expandafter\noexpand\csname#2\string#1\endcsname
9247 ###1\noexpand\@empty\noexpand\@text@composite
9248 {##1}%
9249 }%
9250 }%
9251 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9252 \fi
9253 \expandafter\def\csname\expandafter\string\csname
9254 #2\endcsname\string#1-\string#3\endcsname{#4}

```



```

9255 \else
9256 \errhelp{Your command will be ignored, type <return> to proceed}%
9257 \errmessage{\string\DeclareTextCompositeCommand\space used on
9258 \inappropriate command \protect#1}
9259 \fi
9260 }
9261 \def\@text@composite#1#2#3\@text@composite{%
9262 \expandafter\@text@composite@x
9263 \csname\string#1-\string#2\endcsname
9264 }
9265 \def\@text@composite@x#1#2{%
9266 \ifx#1\relax
9267 #2%
9268 \else
9269 #1%
9270 \fi
9271 }
9272 %
9273 \def\@strip@args#1:#2-#3\@strip@args{#2}
9274 \def\DeclareTextComposite#1#2#3#4{%
9275 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
9276 \bgroup
9277 \lccode`\@=#4%
9278 \lowercase{%
9279 \egroup
9280 \reserved@a \@%
9281 }%
9282 }
9283 %
9284 \def\UseTextSymbol#1#2{#2}
9285 \def\UseTextAccent#1#2#3{}
9286 \def\@use@text@encoding#1{}
9287 \def\DeclareTextSymbolDefault#1#2{%
9288 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9289 }
9290 \def\DeclareTextAccentDefault#1#2{%
9291 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9292 }
9293 \def\cf@encoding{OT1}

```

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

```

9294 \DeclareTextAccent{"}{OT1}{127}
9295 \DeclareTextAccent{'}{OT1}{19}
9296 \DeclareTextAccent{^}{OT1}{94}
9297 \DeclareTextAccent{\`}{OT1}{18}
9298 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9299 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9300 \DeclareTextSymbol{\textquotedblright}{OT1}{'\'}
9301 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9302 \DeclareTextSymbol{\textquoteright}{OT1}{''}
9303 \DeclareTextSymbol{\i}{OT1}{16}
9304 \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`.

```

9305 \ifx\scriptsize\undefined
9306 \let\scriptsize\sevenrm
9307 \fi

```

And a few more “dummy” definitions.

```

9308 \def\language{english}%

```

```

9309 \let\bbl@opt@shorthands\@nnil
9310 \def\bbl@ifshorthand#1#2#3{#2}%
9311 \let\bbl@language@opts\@empty
9312 \let\bbl@provide@locale\relax
9313 \ifx\babeloptionstrings\undefined
9314   \let\bbl@opt@strings\@nnil
9315 \else
9316   \let\bbl@opt@strings\babeloptionstrings
9317 \fi
9318 \def\BabelStringsDefault{generic}
9319 \def\bbl@tempa{normal}
9320 \ifx\babeloptionmath\bbl@tempa
9321   \def\bbl@mathnormal{\noexpand\textormath}
9322 \fi
9323 \def\AfterBabelLanguage#1#2{}
9324 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
9325 \let\bbl@afterlang\relax
9326 \def\bbl@opt@safe{BR}
9327 \ifx\@uclclist\undefined\let\@uclclist\@empty\fi
9328 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
9329 \expandafter\newif\csname ifbbl@single\endcsname
9330 \chardef\bbl@bidimode\z@
9331 <</Emulate LaTeX>

A proxy file:

9332 <*\plain>
9333 \input babel.def
9334 </\plain>

```

15. Acknowledgements

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

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

Barbara Beeton has helped in improving the manual.

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

References

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

- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).