

Babel

Code

Version 26.5
2026/03/24

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TeX

LuaTeX

pdfTeX

XeTeX

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	TeX: babel.sty (start)	8
3.3	base	10
3.4	key=value options and other general option	10
3.5	Post-process some options	12
3.6	Plain: babel.def (start)	13
4	babel.sty and babel.def (common)	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	More on selection	24
4.4	Short tags	25
4.5	Compatibility with language.def	25
4.6	Hooks	26
4.7	Setting up language files	27
4.8	Shorthands	29
4.9	Language attributes	38
4.10	Support for saving and redefining macros	40
4.11	French spacing	41
4.12	Hyphens	42
4.13	Multiencoding strings	44
4.14	Tailor captions	48
4.15	Making glyphs available	49
4.15.1	Quotation marks	49
4.15.2	Letters	51
4.15.3	Shorthands for quotation marks	52
4.15.4	Umlauts and tremas	52
4.16	Layout	54
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	74
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	89
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	113
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	128
10.10	Automatic fonts and ids switching	128
10.11	Bidi	135
10.12	Layout	137
10.13	Lua: transforms	147
10.14	Lua: Auto bidi with basic and basic-r	157
11	Data for CJK	168
12	The ‘nil’ language	168
13	Calendars	169
13.1	Islamic	170
13.2	Hebrew	171
13.3	Persian	175
13.4	Coptic and Ethiopic	176
13.5	Julian	176
13.6	Buddhist	177
14	Support for Plain T_EX (plain.def)	178
14.1	Not renaming hyphen.tex	178
14.2	Emulating some L ^A T _E X features	179
14.3	General tools	179
14.4	Encoding related macros	183
15	Acknowledgements	186

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=26.5>>
2 <<date=2026/03/24>>
```

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@cl#1{\csname bbl@#1@languagename\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`, `\langle.` for `\noexpand` applied to a built macro name (which does not define the macro if undefined to `\relax`, because it is created locally), and `\[. . .]` for one-level expansion (where `. . .` is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

155 \def\bb@ifsamestring#1#2{%
156   \begingroup
157     \protected@edef\bb@tempb{#1}%
158     \edef\bb@tempb{\expandafter\strip@prefix\meaning\bb@tempb}%
159   \protected@edef\bb@tempc{#2}%
160   \edef\bb@tempc{\expandafter\strip@prefix\meaning\bb@tempc}%
161   \ifx\bb@tempb\bb@tempc
162     \aftergroup\@firstoftwo
163   \else
164     \aftergroup\@secondoftwo
165   \fi
166 \endgroup}
167 \chardef\bb@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\bb@bsphack{%
178   \ifhmode
179     \hskip\z@skip
180     \def\bb@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
181   \else
182     \let\bb@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\bb@cased{%
185   \ifx\oe\OE
186     \expandafter\in@\expandafter
187     {\expandafter\OE\expandafter}\expandafter{\oe}%
188   \ifin@
189     \bb@afterelse\expandafter\MakeUppercase
190   \else
191     \bb@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\bb@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
197   \toks@\expandafter\expandafter\expandafter{%
198     \csname extras\languagename\endcsname}%
199   \bb@exp{\in@{#1}}{\the\toks@}}%
200 \ifin@\else
201   \@temptokena{#2}%
202   \edef\bb@tempc{\the\@temptokena\the\toks@}%
203   \toks@\expandafter{\bb@tempc#3}%
204   \expandafter\edef\csname extras\languagename\endcsname{\the\toks@}%
205   \fi}
206 <</Basic macros>>

```

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


```

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}
244 % Temporary:
245 \newif\ifBabelDebugGerman
246 \@ifpackagewith{babel}{debug-german}
247 {\BabelDebugGermantrue}
248 {\BabelDebugGermanfalse}

```

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

```

249 \def\bbl@error#1{% Implicit #2#3#4
250 \begingroup
251 \catcode`\=0 \catcode`\==12 \catcode`\`=12
252 \input errbabel.def
253 \endgroup
254 \bbl@error{#1}}
255 \def\bbl@warning#1{%
256 \begingroup
257 \def\{\MessageBreak}%
258 \PackageWarning{babel}{#1}%
259 \endgroup}
260 \def\bbl@infowarn#1{%
261 \begingroup
262 \def\{\MessageBreak}%
263 \PackageNote{babel}{#1}%
264 \endgroup}
265 \def\bbl@info#1{%
266 \begingroup
267 \def\{\MessageBreak}%
268 \PackageInfo{babel}{#1}%
269 \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.

```

270 <@Basic macros@>
271 \@ifpackagewith{babel}{silent}
272 {\let\bbl@info@gobble
273 \let\bbl@infowarn@gobble
274 \let\bbl@warning@gobble}
275 {}
276 %
277 \def\AfterBabelLanguage#1{%
278 \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.

```

279 \ifx\bbl@languages@undefined\else
280 \begingroup
281 \catcode`\^^I=12
282 \@ifpackagewith{babel}{showlanguages}{%
283 \begingroup
284 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
285 \wlog{<*languages>}%
286 \bbl@languages
287 \wlog{</languages>}%
288 \endgroup}}
289 \endgroup
290 \def\bbl@elt#1#2#3#4{%
291 \ifnum#2=\z@
292 \gdef\bbl@nulllanguage{#1}%
293 \def\bbl@elt##1##2##3##4{%
294 \fi}%
295 \bbl@languages
296 \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`.

```
297 \bbl@trace{Defining option 'base'}
298 \@ifpackagewith{babel}{base}{%
299   \let\bbl@onlyswitch\@empty
300   \let\bbl@provide@locale\relax
301   \input babel.def
302   \let\bbl@onlyswitch\@undefined
303   \ifx\directlua\@undefined
304     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
305   \else
306     \input luabel.def
307     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
308   \fi
309   \DeclareOption{base}{}%
310   \DeclareOption{showlanguages}{}%
311   \ProcessOptions
312   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
313   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
314   \global\let\@ifl@ter@@\@ifl@ter
315   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
316   \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`.

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

```

347 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
348 \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.

```

349 \DeclareOption{KeepShorthandsActive}{}
350 \DeclareOption{activeacute}{}
351 \DeclareOption{activegrave}{}
352 \DeclareOption{debug}{}
353 \DeclareOption{debug-german}{} % Temporary
354 \DeclareOption{noconfigs}{}
355 \DeclareOption{showlanguages}{}
356 \DeclareOption{silent}{}
357 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
358 \chardef\bbl@iniflag\z@
359 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main = 1
360 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % second = 2
361 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % second + main
362 \chardef\bbl@ldfflag\z@
363 \DeclareOption{provide=!}{\chardef\bbl@ldfflag\@ne} % main = 1
364 \DeclareOption{provide+=!}{\chardef\bbl@ldfflag\tw@} % second = 2
365 \DeclareOption{provide*=!}{\chardef\bbl@ldfflag\thr@@} % second + main
366 % Don't use. Experimental.
367 \newif\ifbbl@single
368 \DeclareOption{selectors=off}{\bbl@singletrue}
369 <@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.

```

370 \let\bbl@opt@shorthands\@nnil
371 \let\bbl@opt@config\@nnil
372 \let\bbl@opt@main\@nnil
373 \let\bbl@opt@headfoot\@nnil
374 \let\bbl@opt@layout\@nnil
375 \let\bbl@opt@provide\@nnil

```

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

```

376 \def\bbl@tempa#1=#2\bbl@tempa{%
377 \bbl@csarg\ifx{opt#1}\@nnil
378 \bbl@csarg\edef{opt#1}{#2}%
379 \else
380 \bbl@error{bad-package-option}{#1}{#2}{}%
381 \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.

```

382 \let\bbl@language@opts\@empty
383 \DeclareOption*{%
384 \bbl@xin@{\string=}{\CurrentOption}%
385 \ifin@
386 \expandafter\bbl@tempa\CurrentOption\bbl@tempa
387 \else
388 \bbl@add@list\bbl@language@opts{\CurrentOption}%
389 \fi}

```

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

```

390 \ProcessOptions*

```

3.5. Post-process some options

```

391 \ifx\babel@opt@provide\@nnil
392 \let\babel@opt@provide\@empty %%% MOVE above
393 \else
394 \chardef\babel@iniflag\@ne
395 \babel@exp{\babel@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
396 \in@{,provide,}{,#1,}%
397 \ifin@
398 \def\babel@opt@provide{#2}%
399 \fi}
400 \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 `\babel@ifshorthand` is always true, and it is always false if `shorthands` is empty. Also, some code makes sense only with `shorthands=...`

```

401 \babel@trace{Conditional loading of shorthands}
402 \def\babel@sh@string#1{%
403 \ifx#1\@empty\else
404 \ifx#1t\string~%
405 \else\ifx#1c\string,%
406 \else\string#1%
407 \fi\fi
408 \expandafter\babel@sh@string
409 \fi}
410 \ifx\babel@opt@shorthands\@nnil
411 \def\babel@ifshorthand#1#2#3{#2}%
412 \else\ifx\babel@opt@shorthands\@empty
413 \def\babel@ifshorthand#1#2#3{#3}%
414 \else

```

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

```

415 \def\babel@ifshorthand#1{%
416 \babel@xin{\string#1}{\babel@opt@shorthands}%
417 \ifin@
418 \expandafter\@firstoftwo
419 \else
420 \expandafter\@secondoftwo
421 \fi}

```

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

```

422 \edef\babel@opt@shorthands{%
423 \expandafter\babel@sh@string\babel@opt@shorthands\@empty}%

```

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

```

424 \babel@ifshorthand{'}%
425 {\PassOptionsToPackage{activeacute}{babel}}{}
426 \babel@ifshorthand{`}%
427 {\PassOptionsToPackage{activegrave}{babel}}{}
428 \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.

```

429 \ifx\babel@opt@headfoot\@nnil\else
430 \g@addto@macro\@resetactivechars{%
431 \set@typeset@protect
432 \expandafter\select@language@x\expandafter{\babel@opt@headfoot}%
433 \let\protect\noexpand}
434 \fi

```

For the option `safe` we use a different approach – `\babel@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`.

```

435 \ifx\babel@opt@safe\@undefined

```

```

436 \def\bblopt@safe{BR}
437 % \let\bblopt@safe\@empty % Pending of \cite
438 \fi

```

For layout an auxiliary macro is provided, available for packages and language styles.
Optimization: if there is no layout, just do nothing.

```

439 \bblopt@trace{Defining IfBabelLayout}
440 \ifx\bblopt@layout\@nnil
441 \newcommand\IfBabelLayout[3]{#3}%
442 \else
443 \bblopt@exp{\bblopt@forkv{\@nameuse{\raw@opt@babel.sty}}}{%
444 \in{, layout,}{, #1,}%
445 \ifin@
446 \def\bblopt@layout{#2}%
447 \bblopt@replace\bblopt@layout{ }{.}%
448 \fi}
449 \newcommand\IfBabelLayout[1]{%
450 \@expandtwoargs\in{.#1.}{.\bblopt@layout.}%
451 \ifin@
452 \expandafter\@firstoftwo
453 \else
454 \expandafter\@secondoftwo
455 \fi}
456 \fi
457 </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.

```

458 < *core >
459 \ifx\ldf@quit\@undefined\else
460 \endinput\fi % Same line!
461 <@Make sure ProvidesFile is defined@>
462 \ProvidesFile{babel.def}[<@date@> v<@version@> Babel common definitions]
463 \ifx\AtBeginDocument\@undefined
464 <@Emulate LaTeX@>
465 \fi
466 <@Basic macros@>
467 </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)

```

468 < *package | core >
469 \def\bblopt@version{<@version@>}
470 \def\bblopt@date{<@date@>}
471 <@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.

```

472 \def\adddialect#1#2{%
473 \global\chardef#1#2\relax
474 \bblopt@usehooks{adddialect}{{#1}{#2}}%
475 \begingroup
476 \count@#1\relax
477 \def\bblopt@elt##1##2##3##4{%
478 \ifnum\count@=#2\relax
479 \edef\bblopt@tempa{\expandafter\@gobbletwo\string#1}%
480 \bblopt@info{Hyphen rules for '\expandafter\@gobble\bblopt@tempa'}

```

```

481         set to \expandafter\string\csname l@##1\endcsname\%
482         (\string\language\thecount@). Reported}%
483     \def\bbl@elt####1####2####3####4{%
484         \fi}%
485     \bbl@cs{languages}%
486 \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.

```

487 \def\bbl@fixname#1{%
488     \begingroup
489     \def\bbl@tempe{l@}%
490     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
491     \bbl@tempd
492     {\lowercase\expandafter{\bbl@tempd}%
493     {\uppercase\expandafter{\bbl@tempd}%
494     \@empty
495     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
496     \uppercase\expandafter{\bbl@tempd}}}%
497     {\edef\bbl@tempd{\def\noexpand#1{#1}}%
498     \lowercase\expandafter{\bbl@tempd}}}%
499     \@empty
500     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
501     \bbl@tempd
502     \bbl@exp{\bbl@usehooks{language}{\language}{#1}}
503 \def\bbl@iflanguage#1{%
504     \@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.

```

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

```

```

531 \ifx\bb@bcp\relax
532 \IfFileExists{babel-\bb@tempa-\bb@tempc.ini}%
533 {\edef\bb@bcp{\bb@tempa-\bb@tempc}}}%
534 {}%
535 \fi
536 \ifx\bb@bcp\relax
537 \IfFileExists{babel-\bb@tempa-\bb@tempc.ini}%
538 {\edef\bb@bcp{\bb@tempa-\bb@tempc}}}%
539 {}%
540 \fi
541 \ifx\bb@bcp\relax
542 \IfFileExists{babel-\bb@tempa.ini}{\let\bb@bcp\bb@tempa}}}%
543 \fi
544 \fi\fi}
545 \let\bb@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.

```

546 \def\iflanguage#1{%
547 \bb@iflanguage{#1}{%
548 \ifnum\csname l@#1\endcsname=\language
549 \expandafter\@firstoftwo
550 \else
551 \expandafter\@secondoftwo
552 \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.

```

553 \let\bb@select@type\z@
554 \edef\selectlanguage{%
555 \noexpand\protect
556 \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`.

```
557 \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.

```
558 \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.

\bb@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 `\bb@pop@language` to be executed at the end of the group. It calls `\bb@set@language` with the name of the current language as its argument.

\bb@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 `\bb@language@stack` and initially empty.

```
559 \def\bb@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:

```
560 \def\bbl@push@language{%
561   \ifx\languagename\undefined\else
562     \ifx\currentgrouplevel\undefined
563       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
564     \else
565       \ifnum\currentgrouplevel=\z@
566         \xdef\bbl@language@stack{\languagename+}%
567       \else
568         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
569       \fi
570     \fi
571 \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 `\languagename`. For this we first define a helper function.

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

```
572 \def\bbl@pop@lang#1+#2\@@{%
573   \edef\languagename{#1}%
574   \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).

```
575 \let\bbl@ifrestoring\@secondoftwo
576 \def\bbl@pop@language{%
577   \expandafter\bbl@pop@lang\bbl@language@stack\@@
578   \let\bbl@ifrestoring\@firstoftwo
579   \expandafter\bbl@set@language\expandafter{\languagename}%
580   \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).

```
581 \chardef\localeid\z@
582 \gdef\bbl@id@last{0} % No real need for a new counter
583 \def\bbl@id@assign{%
584   \bbl@ifunset{bbl@id@\languagename}%
585     {\count@bbl@id@last\relax
586      \advance\count@\@ne
587      \global\bbl@csarg\chardef{id@\languagename}\count@
588      \xdef\bbl@id@last{\the\count@}%
589      \ifcase\bbl@engine\or
590        \directlua{
591          Babel.locale_props[\bbl@id@last] = {}
592          Babel.locale_props[\bbl@id@last].name = '\languagename'
593          Babel.locale_props[\bbl@id@last].vars = {}
594        }%
595      \fi}%
596   }%
597   \chardef\localeid\bbl@c{l{id@}}
```

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

```

598 \let\bbl@select@opts@empty
599 \expandafter\def\csname selectlanguage \endcsname{%
600 \ifnextchar[\bbl@select@s{\bbl@select@s[]}}
601 \def\bbl@select@s[#1]#2{%
602 \def\bbl@select@opts{#1}%
603 \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\tw@fi
604 \bbl@push@language
605 \aftergroup\bbl@pop@language
606 \bbl@set@language{#2}}
607 \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 `\languagename` 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).

```

608 \def\BabelContentsFiles{toc,lof,lot}
609 \def\bbl@set@language#1{% from selectlanguage, pop@
610 % The old buggy way. Preserved for compatibility, but simplified
611 \edef\languagename{\expandafter\string#1\@empty}%
612 \select@language{\languagename}%
613 \bbl@xin@{,main,},{,\bbl@select@opts,}%
614 \ifin@
615 \let\bbl@main@language\localename
616 \let\mainlocalename\localename
617 \fi
618 \let\bbl@select@opts@empty
619 % write to aux files
620 \expandafter\ifx\csname date\languagename\endcsname\relax\else
621 \if@filesw
622 \bbl@xin@{,nofiles,},{,\bbl@select@opts,}%
623 \ifin@\else
624 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
625 \bbl@savelastskip
626 \protected@write\@auxout{\string\babel@aux{\bbl@auxname}{}}%
627 \bbl@restorelastskip
628 \fi
629 \bbl@usehooks{write}{}%
630 \fi
631 \fi}
632 \fi}
633 %
634 \let\bbl@restorelastskip\relax
635 \let\bbl@savelastskip\relax
636 %
637 \def\select@language#1{% from set@, babel@aux, babel@toc
638 \ifx\bbl@select@name\@empty
639 \def\bbl@select@name{select}%
640 \fi
641 % set hymap
642 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
643 % set name (when coming from babel@aux)
644 \edef\languagename{#1}%
645 \bbl@fixname\languagename
646 % define \localename when coming from set@, with a trick
647 \ifx\scantokens\@undefined

```

```

648 \def\localename{??}%
649 \else
650 \bbl@exp{\scantokens{\def\localename{\language}\noexpand}\relax}%
651 \fi
652 \bbl@provide@locale
653 \bbl@iflanguage\language{%
654 \let\bbl@select@type\z@
655 \expandafter\bbl@switch\expandafter{\language}}
656 \def\babel@aux#1#2{%
657 \select@language{#1}%
658 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
659 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}%
660 \def\babel@toc#1#2{%
661 \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 `\select language`, 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`.

```

662 \newif\ifbbl@usedategroup
663 \let\bbl@savextras\@empty
664 \def\bbl@switch#1{% from select@, foreign@
665 % restore
666 \originalTeX
667 \expandafter\def\expandafter\originalTeX\expandafter{%
668 \csname noextras#1\endcsname
669 \let\originalTeX\@empty
670 \babel@beginsave}%
671 \bbl@usehooks{afterreset}}}%
672 \languageshorthands{none}%
673 % set the locale id
674 \bbl@id@assign
675 % switch captions, date
676 \bbl@bsphack
677 \ifcase\bbl@select@type
678 \csname captions#1\endcsname\relax
679 \csname date#1\endcsname\relax
680 \else
681 \bbl@xin@{,captions,},{, \bbl@select@opts,}%
682 \ifin@
683 \csname captions#1\endcsname\relax
684 \fi
685 \bbl@xin@{,date,},{, \bbl@select@opts,}%
686 \ifin@ % if \foreign... within \<language>date
687 \csname date#1\endcsname\relax
688 \fi
689 \fi
690 \bbl@esphack
691 % switch extras
692 \csname bbl@preextras@#1\endcsname
693 \bbl@usehooks{beforeextras}}}%
694 \csname extras#1\endcsname\relax
695 \bbl@usehooks{afterextras}}}%

```

```

696 % > babel-ensure
697 % > babel-sh-<short>
698 % > babel-bidi
699 % > babel-fontspec
700 \let\bbbl@savedextras\@empty
701 % hyphenation - case mapping
702 \ifcase\bbbl@opt@hyphenmap\or
703   \def\BabelLower##1##2{\lccode##1=##2\relax}%
704   \ifnum\bbbl@hymapsel>4\else
705     \csname\languagename @bbbl@hyphenmap\endcsname
706     \fi
707   \chardef\bbbl@opt@hyphenmap\z@
708 \else
709   \ifnum\bbbl@hymapsel>\bbbl@opt@hyphenmap\else
710     \csname\languagename @bbbl@hyphenmap\endcsname
711     \fi
712 \fi
713 \let\bbbl@hymapsel\@cclv
714 % hyphenation - select rules
715 \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
716   \edef\bbbl@tempa{u}%
717 \else
718   \edef\bbbl@tempa{\bbbl@cl{lbrk}}%
719 \fi
720 % linebreaking - handle u, e, k (v in the future)
721 \bbbl@xin@{/u}{/\bbbl@tempa}%
722 \ifin@ \else \bbbl@xin@{/e}{/\bbbl@tempa} \fi % elongated forms
723 \ifin@ \else \bbbl@xin@{/k}{/\bbbl@tempa} \fi % only kashida
724 \ifin@ \else \bbbl@xin@{/p}{/\bbbl@tempa} \fi % padding (e.g., Tibetan)
725 \ifin@ \else \bbbl@xin@{/v}{/\bbbl@tempa} \fi % variable font
726 % hyphenation - save mins
727 \babel@savevariable\lefthyphenmin
728 \babel@savevariable\righthyphenmin
729 \ifnum\bbbl@engine=\@ne
730   \babel@savevariable\hyphenationmin
731 \fi
732 \ifin@
733   % unhyphenated/kashida/elongated/padding = allow stretching
734   \language\l@unhyphenated
735   \babel@savevariable\emergencystretch
736   \emergencystretch\maxdimen
737   \babel@savevariable\hbadness
738   \hbadness\@M
739 \else
740   % other = select patterns
741   \bbbl@patterns{#1}%
742 \fi
743 % hyphenation - set mins
744 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
745   \set@hyphenmins\tw@\thr@@\relax
746   \@nameuse{bbbl@hyphenmins@}%
747 \else
748   \expandafter\expandafter\expandafter\set@hyphenmins
749     \csname #1hyphenmins\endcsname\relax
750 \fi
751 \@nameuse{bbbl@hyphenmins@}%
752 \@nameuse{bbbl@hyphenmins@\languagename}%
753 \@nameuse{bbbl@hyphenatmin@}%
754 \@nameuse{bbbl@hyphenatmin@\languagename}%
755 \let\bbbl@selectortname\@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.

```
756 \edef\otherlanguage{%
757   \noexpand\protect
758   \expandafter\noexpand\csname otherlanguage \endcsname}
759 \expandafter\def\csname otherlanguage \endcsname{%
760   \@ifstar{\@nameuse{otherlanguage*}}\bbl@otherlanguage}
761 \def\bbl@otherlanguage#1{%
762   \def\bbl@selectorname{other}%
763   \ifnum\bbl@hymapsel=\@ccclv\let\bbl@hymapsel\thr@@\fi
764   \csname selectlanguage \endcsname{#1}%
765   \ignorespaces}
```

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

```
766 \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`.

```
767 \expandafter\def\csname otherlanguage*\endcsname{%
768   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}]
769 \def\bbl@otherlanguage@s[#1]#2{%
770   \def\bbl@selectorname{other*}%
771   \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
772   \def\bbl@select@opts{#1}%
773   \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”.

```
774 \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.

```
775 \providecommand\bbl@beforeforeign{}
776 \edef\foreignlanguage{%
777   \noexpand\protect
778   \expandafter\noexpand\csname foreignlanguage \endcsname}
779 \expandafter\def\csname foreignlanguage \endcsname{%
780   \@ifstar\bbl@foreign@s\bbl@foreign@x}
781 \providecommand\bbl@foreign@x[3][[]]{%
782   \begingroup
783   \def\bbl@selectorname{foreign}%
784   \def\bbl@select@opts{#1}%
785   \let\BabelText\@firstofone
```

```

786 \bbl@beforeforeign
787 \foreign@language{#2}%
788 \bbl@usehooks{foreign}{}%
789 \BabelText{#3}% Now in horizontal mode!
790 \endgroup}
791 \def\bbl@foreign@s#1#2{%
792 \begingroup
793 {\par}%
794 \def\bbl@selectorname{foreign*}%
795 \let\bbl@select@opts@empty
796 \let\BabelText\@firstofone
797 \foreign@language{#1}%
798 \bbl@usehooks{foreign*}{}%
799 \bbl@dirparastext
800 \BabelText{#2}% Still in vertical mode!
801 {\par}%
802 \endgroup}
803 \providecommand\BabelWrapText[1]{%
804 \def\bbl@tempa{\def\BabelText###1}%
805 \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`.

```

806 \def\foreign@language#1{%
807 % set name
808 \edef\languagename{#1}%
809 \ifbbl@usedategroup
810 \bbl@add\bbl@select@opts{,date,}%
811 \bbl@usedategroupfalse
812 \fi
813 \bbl@fixname\languagename
814 \let\localename\languagename
815 \bbl@provide@locale
816 \bbl@iflanguage\languagename{%
817 \let\bbl@select@type@ne
818 \expandafter\bbl@switch\expandafter{\languagename}}

```

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

```

819 \def\IfBabelSelectorTF#1{%
820 \bbl@xin@{\bbl@selectorname,}{,\zap@space#1 \@empty,}%
821 \ifin@
822 \expandafter\@firstoftwo
823 \else
824 \expandafter\@secondoftwo
825 \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.

```

826 \let\bbl@hyphlist\@empty
827 \let\bbl@hyphenation@\relax
828 \let\bbl@pttnlist\@empty
829 \let\bbl@patterns@\relax
830 \let\bbl@hymapsel=\cclv
831 \def\bbl@patterns#1{%
832 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax

```

```

833     \csname l@#1\endcsname
834     \edef\bb1@tempa{#1}%
835     \else
836     \csname l@#1:\f@encoding\endcsname
837     \edef\bb1@tempa{#1:\f@encoding}%
838     \fi
839 \@expandtwoargs\bb1@usehooks{patterns}{#1}{\bb1@tempa}%
840 % > luatex
841 \ifundefined{bb1@hyphenation@}{% Can be \relax!
842 \begingroup
843 \bb1@xin{, \number\language, }{, \bb1@hyphlist}%
844 \ifin@else
845 \@expandtwoargs\bb1@usehooks{hyphenation}{#1}{\bb1@tempa}%
846 \hyphenation{%
847 \bb1@hyphenation@
848 \ifundefined{bb1@hyphenation@#1}%
849 \@empty
850 {\space\csname bb1@hyphenation@#1\endcsname}}%
851 \xdef\bb1@hyphlist{\bb1@hyphlist\number\language,}%
852 \fi
853 \endgroup}}

```

hyphenrules It can be used to select *just* the hyphenation rules. It does *not* change `\languagename` 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*`.

```

854 \def\hyphenrules#1{%
855 \edef\bb1@tempf{#1}%
856 \bb1@fixname\bb1@tempf
857 \bb1@iflanguage\bb1@tempf{%
858 \expandafter\bb1@patterns\expandafter{\bb1@tempf}%
859 \ifx\languageshortands\@undefined\else
860 \languageshortands{none}%
861 \fi
862 \expandafter\ifx\csname\bb1@tempf hyphenmins\endcsname\relax
863 \set@hyphenmins\tw@\thr@@\relax
864 \else
865 \expandafter\expandafter\expandafter\set@hyphenmins
866 \csname\bb1@tempf hyphenmins\endcsname\relax
867 \fi}}
868 \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.

```

869 \def\providehyphenmins#1#2{%
870 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
871 \namedef{#1hyphenmins}{#2}%
872 \fi}

```

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

```

873 \def\set@hyphenmins#1#2{%
874 \lefthyphenmin#1\relax
875 \righthyphenmin#2\relax}

```

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

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

```

876 \ifx\ProvidesFile\@undefined

```

```

877 \def\ProvidesLanguage#1[#2 #3 #4]{%
878   \wlog{Language: #1 #4 #3 <#2>}%
879   }
880 \else
881   \def\ProvidesLanguage#1{%
882     \begingroup
883       \catcode\ 10 %
884       \@makeother\/%
885       \@ifnextchar[%]
886         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
887   \def\@provideslanguage#1[#2]{%
888     \wlog{Language: #1 #2}%
889     \expandafter\xdef\curname ver@#1.ldf\endcsname{#2}%
890     \endgroup}
891 \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`.

```
892 \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.

```
893 \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’:

```

894 \providecommand\setlocale{\bbl@error{not-yet-available}}{}{}
895 \let\uselocale\setlocale
896 \let\locale\setlocale
897 \let\selectlocale\setlocale
898 \let\textlocale\setlocale
899 \let\textlanguage\setlocale
900 \let\languagetext\setlocale

```

4.2. Errors

@nolanerr

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

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

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

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

```

901 \edef\bbl@nulllanguage{\string\language=0}
902 \def\bbl@nocaption{\protect\bbl@nocaption@i}
903 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
904   \global\@namedef{#2}{\textbf{?#1?}}%
905   \@nameuse{#2}%
906   \edef\bbl@tempa{#1}%
907   \bbl@sreplace\bbl@tempa{name}}}%
908   \bbl@sreplace\bbl@tempa{NAME}}}%
909   \bbl@warning{%
910     \@backslashchar#1 not set for '\languagename'. Please,\\%
911     define it after the language has been loaded\\%
912     (typically in the preamble) with:\\%
913     \string\setlocalecaption{\languagename}{\bbl@tempa}{.}\\%
914     Feel free to contribute on github.com/latex3/babel.\\%
915     Reported}}

```



```

916 \def\bb@tentative{\protect\bb@tentative@i}
917 \def\bb@tentative@i#1{%
918   \bb@warning{%
919     Some functions for '#1' are tentative.\\%
920     They might not work as expected and their behavior\\%
921     could change in the future.\\%
922     Reported}}
923 \def\@nolanerr#1{\bb@error{undefined-language}{#1}{}}
924 \def\@nopatterns#1{%
925   \bb@warning
926     {No hyphenation patterns were preloaded for\\%
927     the language '#1' into the format.\\%
928     Please, configure your TeX system to add them and\\%
929     rebuild the format. Now I will use the patterns\\%
930     preloaded for \bb@nulllanguage\space instead}}
931 \let\bb@usehooks\@gobbletwo

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

932 \ifx\bb@onlyswitch\@empty\endinput\fi

```

4.3. More on selection

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

```

933 \bb@trace{Defining babelensure}
934 \newcommand\babelensure[2][]{%
935   \AddBabelHook{babel-ensure}{afterextras}{%
936     \ifcase\bb@select@type
937       \bb@cl{e}%
938       \fi}%
939   \begingroup
940     \let\bb@ens@include\@empty
941     \let\bb@ens@exclude\@empty
942     \def\bb@ens@fontenc{\relax}%
943     \def\bb@tempb##1{%
944       \ifx\@empty##1\else\noexpand##1\expandafter\bb@tempb\fi}%
945     \edef\bb@tempa{\bb@tempb#1\@empty}%
946     \def\bb@tempb##1=##2\@{\@namedef{\bb@ens@##1}{##2}}%
947     \bb@foreach\bb@tempa{\bb@tempb##1\@}%
948     \def\bb@tempc{\bb@ensure}%
949     \expandafter\bb@add\expandafter\bb@tempc\expandafter{%
950       \expandafter{\bb@ens@include}}%
951     \expandafter\bb@add\expandafter\bb@tempc\expandafter{%
952       \expandafter{\bb@ens@exclude}}%
953     \toks@\expandafter{\bb@tempc}%
954     \bb@exp{%
955     \endgroup
956     \def\<bb@e@#2>{\the\toks@{\bb@ens@fontenc}}}%
957 \def\bb@ensure#1#2#3% 1: include 2: exclude 3: fontenc
958 \def\bb@tempb##1{% elt for (excluding) \bb@captionslist list
959   \ifx##1\undefined % 3.32 - Don't assume the macro exists
960     \edef##1{\noexpand\bb@nocaption
961       {\bb@stripslash##1}{\language\bb@stripslash##1}}%
962     \fi
963     \ifx##1\@empty\else

```

```

964 \in{##1}{#2}%
965 \ifin@else
966 \bbl@ifunset{bbl@ensure@\language}%
967 {\bbl@exp{%
968 \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
969 \\\foreignlanguage{\language}%
970 {\ifx\relax#3\else
971 \\\fontencoding{#3}\selectfont
972 \fi
973 #####1}}}%
974 {}%
975 \toks@expandafter{##1}%
976 \edef##1{%
977 \bbl@csarg\noexpand{ensure@\language}%
978 {\the\toks@}}%
979 \fi
980 \expandafter\bbl@tempb
981 \fi}%
982 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
983 \def\bbl@tempa##1{% elt for include list
984 \ifx##1\@empty\else
985 \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
986 \ifin@else
987 \bbl@tempb##1\@empty
988 \fi
989 \expandafter\bbl@tempa
990 \fi}%
991 \bbl@tempa#1\@empty}
992 \def\bbl@captionslist{%
993 \prefacename\refname\abstractname\bibname\chaptername\appendixname
994 \contentsname\listfigurename\listtablename\indexname\figurename
995 \tablename\partname\enclname\ccname\headtoname\pagename\seename
996 \alsoname\proofname\glossaryname}

```

4.4. Short tags

\bbltags 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.

```

997 \bbl@trace{Short tags}
998 \newcommand\bbltags[1]{%
999 \edef\bbl@tempa{\zap@space#1 \@empty}%
1000 \def\bbl@tempb##1=##2\@{%
1001 \edef\bbl@tempc{%
1002 \noexpand\newcommand
1003 \expandafter\noexpand\csname ##1\endcsname{%
1004 \noexpand\protect
1005 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1006 \noexpand\newcommand
1007 \expandafter\noexpand\csname text##1\endcsname{%
1008 \noexpand\foreignlanguage{##2}}}
1009 \bbl@tempc}%
1010 \bbl@for\bbl@tempa\bbl@tempa{%
1011 \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.

```

1012 \bbl@trace{Compatibility with language.def}
1013 \ifx\directlua@undefined\else
1014 \ifx\bbl@luapatterns@undefined
1015 \input luababel.def

```

```

1016 \fi
1017 \fi
1018 \ifx\babel@languages\@undefined
1019 \ifx\directlua\@undefined
1020 \openin1 = language.def
1021 \ifeof1
1022 \closein1
1023 \message{I couldn't find the file language.def}
1024 \else
1025 \closein1
1026 \begingroup
1027 \def\addlanguage#1#2#3#4#5{%
1028 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1029 \global\expandafter\let\csname l@#1\expandafter\endcsname
1030 \csname lang@#1\endcsname
1031 \fi}%
1032 \def\uselanguage#1{%
1033 \input language.def
1034 \endgroup
1035 \fi
1036 \fi
1037 \chardef\l@english\z@
1038 \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.

```

1039 \def\addto#1#2{%
1040 \ifx#1\@undefined
1041 \def#1{#2}%
1042 \else
1043 \ifx#1\relax
1044 \def#1{#2}%
1045 \else
1046 {\toks@\expandafter{#1#2}%
1047 \xdef#1{\the\toks@}}%
1048 \fi
1049 \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. `\babel@usehooks` is the commands used by babel to execute hooks defined for an event.

```

1050 \babel@trace{Hooks}
1051 \newcommand\AddBabelHook[3][]{%
1052 \babel@ifunset{babel@hk@#2}{\EnableBabelHook{#2}}{%
1053 \def\babel@tempa##1,##2=##2,##3\@empty{\def\babel@tempb{##2}}%
1054 \expandafter\babel@tempa\babel@evargs,##3=,\@empty
1055 \babel@ifunset{babel@ev@#2@#3@#1}%
1056 {\babel@csarg\babel@add{ev@#3@#1}{\babel@elth{#2}}}%
1057 {\babel@csarg\let{ev@#2@#3@#1}\relax}%
1058 \babel@csarg\newcommand{ev@#2@#3@#1}{\babel@tempb}}
1059 \newcommand\EnableBabelHook[1]{\babel@csarg\let{hk@#1}\@firstofone}
1060 \newcommand\DisableBabelHook[1]{\babel@csarg\let{hk@#1}\@gobble}
1061 \def\babel@usehooks{\babel@usehooks@lang\languagename}
1062 \def\babel@usehooks@lang#1#2#3{% Test for Plain
1063 \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1064 \def\babel@elth##1{%
1065 \babel@cs{hk@##1}{\babel@cs{ev@##1@#2@#3}}%

```

```

1066 \bbl@cs{ev@#2@}%
1067 \ifx\language\@undefined\else % Test required for Plain (?)
1068 \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1069 \def\bbl@elth##1{%
1070 \bbl@cs{hk###1}{\bbl@cs{ev###1@#2@#1}#3}}%
1071 \bbl@cs{ev@#2@#1}%
1072 \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).

```

1073 \def\bbl@evargs{,% <- don't delete this comma
1074 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1075 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1076 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1077 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1078 beforestart=0,language=2,begindocument=1}
1079 \ifx\NewHook\@undefined\else % Test for Plain (?)
1080 \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1081 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1082 \fi

```

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

```

1083 \providecommand\PassOptionsToLocale[2]{%
1084 \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.

```

1085 \bbl@trace{Macros for setting language files up}
1086 \def\bbl@ldfinit{%
1087 \let\bbl@screset\@empty
1088 \let\BabelStrings\bbl@opt@string
1089 \let\BabelOptions\@empty
1090 \let\BabelLanguages\relax
1091 \ifx\originalTeX\@undefined
1092 \let\originalTeX\@empty
1093 \else
1094 \originalTeX
1095 \fi}
1096 \def\LdfInit#1#2{%
1097 \chardef\atcatcode=\catcode`\@
1098 \catcode`\@=11\relax
1099 \chardef\eqcatcode=\catcode`\=
1100 \catcode`\=12\relax
1101 \@ifpackagewith{babel}{ensureinfo=off}{}}%

```

```

1102   {\ifx\InputIfFileExists\@undefined\else
1103     \bbl@ifunset{bbl@lname@#1}%
1104     {\let\bbl@ensuring@empty % Flag used in babel-serbianc.tex
1105       \def\languagename{#1}%
1106       \bbl@id@assign
1107       \bbl@load@info{#1}}}%
1108     }%
1109   \fi}%
1110 \expandafter\if\expandafter\@backslashchar
1111   \expandafter\@car\string#2\@nil
1112   \ifx#2\@undefined\else
1113     \ldf@quit{#1}%
1114   \fi
1115 \else
1116   \expandafter\ifx\csname#2\endcsname\relax\else
1117     \ldf@quit{#1}%
1118   \fi
1119 \fi
1120 \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.

```

1121 \def\ldf@quit#1{%
1122   \expandafter\main@language\expandafter{#1}%
1123   \catcode`\@=\atcatcode \let\atcatcode\relax
1124   \catcode`\==\eqcatcode \let\eqcatcode\relax
1125   \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.

```

1126 \def\bbl@afterldf{%
1127   \bbl@afterlang
1128   \let\bbl@afterlang\relax
1129   \let\BabelModifiers\relax
1130   \let\bbl@screaset\relax}%
1131 \def\ldf@finish#1{%
1132   \loadlocalcfg{#1}%
1133   \bbl@afterldf
1134   \expandafter\main@language\expandafter{#1}%
1135   \catcode`\@=\atcatcode \let\atcatcode\relax
1136   \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1137 \@onlypreamble\LdfInit
1138 \@onlypreamble\ldf@quit
1139 \@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.

```

1140 \def\main@language#1{%
1141   \def\bbl@main@language{#1}%
1142   \let\languagename\bbl@main@language
1143   \let\localename\bbl@main@language
1144   \let\mainlocalename\bbl@main@language
1145   \bbl@id@assign

```

```

1146 \ifcase\bbl@engine\or
1147   \ifx\setattribute\undefined\else
1148     \setattribute\bbl@attr@locale\localeid
1149   \fi
1150 \fi
1151 \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.

```

1152 \def\bbl@beforestart{%
1153   \def\nolanerr##1{%
1154     \bbl@carg\chardef{l@##1}\z@
1155     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1156   \bbl@usehooks{beforestart}{}%
1157   \global\let\bbl@beforestart\relax
1158 \AtBeginDocument{%
1159   {\@nameuse{bbl@beforestart}}% Group!
1160   \if@filesw
1161     \providecommand\babel@aux[2]{}%
1162     \immediate\write\@mainaux{\unexpanded{%
1163       \providecommand\babel@aux[2]{\global\let\babel@toc@gobbletwo}}}%
1164     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1165   \fi
1166   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1167   \ifbbl@single % must go after the line above.
1168     \renewcommand\selectlanguage[1]{}%
1169     \renewcommand\foreignlanguage[2]{#2}%
1170     \global\let\babel@aux@gobbletwo % Also as flag
1171   \fi}
1172 %
1173 \ifcase\bbl@engine\or
1174   \AtBeginDocument{\pagedir\bodydir}
1175 \fi
```

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

```

1176 \def\select@language@x#1{%
1177   \ifcase\bbl@select@type
1178     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1179   \else
1180     \select@language{#1}%
1181   \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.

```

1182 \bbl@trace{Shorhands}
1183 \def\bbl@withactive#1#2{%
1184   \begingroup
1185     \lccode`~=#2\relax
1186     \lowercase{\endgroup#1~}}
```

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

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

```

1187 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
```

```

1188 \bbl@add\dospecials{do#1}% test @sanitize = \relax, for back. compat.
1189 \bbl@ifunset{@sanitize}{\bbl@add@\sanitize{@makeother#1}}%
1190 \ifx\nfss@catcodes\undefined\else
1191   \begingroup
1192     \catcode`#1\active
1193     \nfss@catcodes
1194     \ifnum\catcode`#1=\active
1195       \endgroup
1196       \bbl@add\nfss@catcodes{@makeother#1}%
1197     \else
1198       \endgroup
1199   \fi
1200 \fi}

```

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

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

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

```

1201 \def\bbl@active@def#1#2#3#4{%
1202   \@namedef{#3#1}{%
1203     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1204     \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1205   \else
1206     \bbl@afterfi\csname#2@sh@#1@\endcsname
1207   \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.

```

1208 \long\@namedef{#3@arg#1}##1{%
1209   \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1210   \bbl@afterelse\csname#4#1\endcsname##1%
1211 \else
1212   \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1213 \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.

```

1214 \def\@initiate@active@char#1{%
1215   \bbl@ifunset{active@char\string#1}%
1216   {\bbl@withactive
1217    {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1218   {}}

```

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).

```

1219 \def\@initiate@active@char#1#2#3{%
1220   \bbl@csarg\edef{oricat#2}{\catcode`#2=\the\catcode`#2\relax}%
1221   \ifx#1\undefined

```

```

1222 \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\undefined}}%
1223 \else
1224 \bbl@csarg\let{oridef@@#2}#1%
1225 \bbl@csarg\edef{oridef@#2}{%
1226 \let\noexpand#1%
1227 \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1228 \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`).

```

1229 \ifx#1#3\relax
1230 \expandafter\let\csname normal@char#2\endcsname#3%
1231 \else
1232 \bbl@info{Making #2 an active character}%
1233 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1234 \@namedef{normal@char#2}{%
1235 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1236 \else
1237 \@namedef{normal@char#2}{#3}%
1238 \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).

```

1239 \bbl@restoreactive{#2}%
1240 \AtBeginDocument{%
1241 \catcode`#2\active
1242 \if@filesw
1243 \immediate\write\@mainaux{\catcode`\string#2\active}%
1244 \fi}%
1245 \expandafter\bbl@add@special\csname#2\endcsname
1246 \catcode`#2\active
1247 \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⟩`).

```

1248 \let\bbl@tempa\@firstoftwo
1249 \if\string^#2%
1250 \def\bbl@tempa{\noexpand\textormath}%
1251 \else
1252 \ifx\bbl@mathnormal\undefined\else
1253 \let\bbl@tempa\bbl@mathnormal
1254 \fi
1255 \fi
1256 \expandafter\edef\csname active@char#2\endcsname{%
1257 \bbl@tempa
1258 {\noexpand\if@safe@actives
1259 \noexpand\expandafter
1260 \expandafter\noexpand\csname normal@char#2\endcsname
1261 \noexpand\else
1262 \noexpand\expandafter
1263 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1264 \noexpand\fi}%
1265 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1266 \bbl@csarg\edef{doactive#2}{%

```



```
1267 \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

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

(where $\backslash\text{active@char}\langle\text{char}\rangle$ is *one* control sequence!).

```
1268 \bbl@csarg\edef{active@#2}{%
1269 \noexpand\active@prefix\noexpand#1%
1270 \expandafter\noexpand\csname active@char#2\endcsname}%
1271 \bbl@csarg\edef{normal@#2}{%
1272 \noexpand\active@prefix\noexpand#1%
1273 \expandafter\noexpand\csname normal@char#2\endcsname}%
1274 \bbl@ncarg\let#1{\bbl@normal@#2}%
```

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

```
1275 \bbl@active@def#2\user@group{user@active}{language@active}%
1276 \bbl@active@def#2\language@group{language@active}{system@active}%
1277 \bbl@active@def#2\system@group{system@active}{normal@char}%
```

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

```
1278 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1279 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1280 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1281 {\expandafter\noexpand\csname user@active#2\endcsname}%
```

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

```
1282 \if\string'#2%
1283 \let\prim@s\bbl@prim@s
1284 \let\active@math@prime#1%
1285 \fi
1286 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}
```

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

```
1287 <<{*More package options}>> ≡
1288 \DeclareOption{math=active}{}
1289 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1290 <</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.

```
1291 \ifpackagewith{babel}{KeepShorthandsActive}%
1292 {\let\bbl@restoreactive@gobble}%
1293 {\def\bbl@restoreactive#1{%
1294 \bbl@exp{%
1295 \\\AfterBabelLanguage\\CurrentOption
1296 {\catcode`#1=\the\catcode`#1\relax}%
1297 \\\AtEndOfPackage
1298 {\catcode`#1=\the\catcode`#1\relax}}}%
1299 \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.

```
1300 \def\bbl@sh@select#1#2{%
1301   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1302   \bbl@afterelse\bbl@scndcs
1303   \else
1304   \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1305   \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.

```
1306 \begingroup
1307 \bbl@ifunset{ifincsname}
1308 {\gdef\active@prefix#1{%
1309   \ifx\protect\@typeset@protect
1310   \else
1311   \ifx\protect\@unexpandable@protect
1312   \noexpand#1%
1313   \else
1314   \protect#1%
1315   \fi
1316   \expandafter\@gobble
1317   \fi}}
1318 {\gdef\active@prefix#1{%
1319   \ifincsname
1320   \string#1%
1321   \expandafter\@gobble
1322   \else
1323   \ifx\protect\@typeset@protect
1324   \else
1325   \ifx\protect\@unexpandable@protect
1326   \noexpand#1%
1327   \else
1328   \protect#1%
1329   \fi
1330   \expandafter\expandafter\expandafter\@gobble
1331   \fi
1332   \fi}}
1333 \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`).

```
1334 \newif\if@safe@actives
1335 \@safe@activesfalse
```

\bbl@restore@actives When the output routine kicks in while the active characters were made “safe” this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them “unsafe” again.

```
1336 \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`.

```
1337 \chardef\bbl@activated\z@
1338 \def\bbl@activate#1{%
1339   \chardef\bbl@activated\@ne
1340   \bbl@withactive{\expandafter\let\expandafter}#1%
1341   \csname bbl@active@string#1\endcsname}
1342 \def\bbl@deactivate#1{%
1343   \chardef\bbl@activated\tw@
1344   \bbl@withactive{\expandafter\let\expandafter}#1%
1345   \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs

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

```
1346 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1347 \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.

```
1348 \def\babel@texpdf#1#2#3#4{%
1349   \ifx\texorpdfstring\undefined
1350     \textormath{#1}{#3}%
1351   \else
1352     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1353     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1354   \fi}
1355 %
1356 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1357 \def\@decl@short#1#2#3\@nil#4{%
1358   \def\bbl@tempa{#3}%
1359   \ifx\bbl@tempa\@empty
1360     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1361     \bbl@ifunset{#1@sh@\string#2@}{}%
1362     {\def\bbl@tempa{#4}%
1363      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1364      \else
1365        \bbl@info
1366          {Redefining #1 shorthand \string#2\}%
1367          in language \CurrentOption}%
1368     \fi}%
1369   \@namedef{#1@sh@\string#2@}{#4}%
1370 \else
1371   \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1372   \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1373   {\def\bbl@tempa{#4}%
1374    \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1375    \else
1376      \bbl@info
1377        {Redefining #1 shorthand \string#2\string#3\}%
1378        in language \CurrentOption}%
1379    \fi}%
1380   \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1381 \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.

```

1382 \def\textormath{%
1383   \ifmmode
1384     \expandafter\@secondoftwo
1385   \else
1386     \expandafter\@firstoftwo
1387   \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’.

```

1388 \def\user@group{user}
1389 \def\language@group{english}
1390 \def\system@group{system}

```

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

```

1391 \def\usesshorthands{%
1392   \@ifstar\bb@useseshs{\bb@useseshx{}}
1393 \def\bb@useseshs#1{%
1394   \bb@useseshx
1395   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1396   {#1}}
1397 \def\bb@useseshx#1#2{%
1398   \bb@ifshorthand{#2}%
1399   {\def\user@group{user}%
1400    \initiate@active@char{#2}%
1401    #1%
1402    \bb@activate{#2}}%
1403   {\bb@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 `\bb@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1404 \def\user@language@group{user@language@group}
1405 \def\bb@set@user@generic#1#2{%
1406   \bb@ifunset{user@generic@active#1}%
1407   {\bb@active@def#1\user@language@group{user@active}{user@generic@active}%
1408    \bb@active@def#1\user@group{user@generic@active}{language@active}%
1409    \expandafter\edef\csname#2@sh@#1@\endcsname{%
1410     \expandafter\noexpand\csname normal@char#1\endcsname}%
1411     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1412      \expandafter\noexpand\csname user@active#1\endcsname}}%
1413   \@empty}
1414 \newcommand\defineshorthand[3][user]{%
1415   \edef\bb@tempa{\zap@space#1 \@empty}%
1416   \bb@for\bb@tempb\bb@tempa{%
1417     \if*\expandafter\@car\bb@tempb\nil
1418       \edef\bb@tempb{user\expandafter\@gobble\bb@tempb}%
1419       \@expandtwoargs
1420       \bb@set@user@generic{\expandafter\string\@car#2\nil}\bb@tempb
1421     \fi
1422     \declare@shorthand{\bb@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.

```

1423 \def\languageshorthands#1{%
1424   \bbl@ifsamestring{none}{#1}{}%
1425   \bbl@once{short-\localename-#1}{%
1426     \bbl@info{'\localename' activates '#1' shorthands.\Reported}}}%
1427   \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`".

```

1428 \def\aliasshorthand#1#2{%
1429   \bbl@ifshorthand{#2}%
1430   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1431     \ifx\document\@notprerr
1432       \@notshorthand{#2}%
1433     \else
1434       \initiate@active@char{#2}%
1435       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1436       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1437       \bbl@activate{#2}%
1438     \fi
1439   \fi}%
1440   {\bbl@error{shorthand-is-off}{#2}{}}

```

\@notshorthand

```

1441 \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.

```

1442 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1443 \DeclareRobustCommand*\shorthandoff{%
1444   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1445 \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.

```

1446 \def\bbl@switch@sh#1#2{%
1447   \ifx#2\@nnil\else
1448     \bbl@ifunset{bbl@active@\string#2}%
1449     {\bbl@error{not-a-shorthand-b}{#2}{}}%
1450     {\ifcase#1%   off, on, off*
1451       \catcode`#2\relax
1452     \or
1453       \catcode`#2\active
1454       \bbl@ifunset{bbl@shdef@\string#2}%
1455       {}%
1456       {\bbl@withactive{\expandafter\let\expandafter}#2%
1457         \csname bbl@shdef@\string#2\endcsname
1458         \bbl@csarg\let{shdef@\string#2}\relax}%
1459     \ifcase\bbl@activated\or
1460       \bbl@activate{#2}%

```

```

1461     \else
1462     \bbl@deactivate{#2}%
1463     \fi
1464     \or
1465     \bbl@ifunset{bbl@shdef@\string#2}%
1466     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1467     {}%
1468     \csname bbl@oricat@\string#2\endcsname
1469     \csname bbl@oridef@\string#2\endcsname
1470     \fi}%
1471     \bbl@afterfi\bbl@switch@sh#1%
1472     \fi}

```

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

```

1473 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1474 \def\bbl@putsh#1{%
1475   \bbl@ifunset{bbl@active@\string#1}%
1476   {\bbl@putsh@i#1@empty\@nnil}%
1477   {\csname bbl@active@\string#1\endcsname}}
1478 \def\bbl@putsh@i#1#2\@nnil{%
1479   \csname\language@group @sh@\string#1@%
1480   \ifx@empty#2\else\string#2\fi\endcsname}
1481 %
1482 \ifx\bbl@opt@shorthands\@nnil\else
1483   \let\bbl@s@initiate@active@char\initiate@active@char
1484   \def\initiate@active@char#1{%
1485     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1486   \let\bbl@s@switch@sh\bbl@switch@sh
1487   \def\bbl@switch@sh#1#2{%
1488     \ifx#2\@nnil\else
1489       \bbl@afterfi
1490       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1491       \fi}
1492   \let\bbl@s@activate\bbl@activate
1493   \def\bbl@activate#1{%
1494     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1495   \let\bbl@s@deactivate\bbl@deactivate
1496   \def\bbl@deactivate#1{%
1497     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1498 \fi

```

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

```

1499 \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.

```

1500 \def\bbl@prim@s{%
1501   \prime\futurelet\@let@token\bbl@pr@m@s}
1502 \def\bbl@if@primes#1#2{%
1503   \ifx#1\@let@token
1504     \expandafter\@firstoftwo
1505   \else\ifx#2\@let@token
1506     \bbl@afterelse\expandafter\@firstoftwo
1507   \else
1508     \bbl@afterfi\expandafter\@secondoftwo
1509   \fi\fi}
1510 \begingroup
1511 \catcode\^=7 \catcode\*=active \lccode\^=\^

```

```

1512 \catcode`\'=12 \catcode`\"=\active \lccode`\"=\`
1513 \lowercase{%
1514   \gdef\bbl@pr@ms{%
1515     \bbl@if@primes" '%
1516     \pr@@s
1517     {\bbl@if@primes*^\pr@@t\egroup}}
1518 \endgroup

```

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

```

1519 \initiate@active@char{~}
1520 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1521 \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.

```

1522 \expandafter\def\csname OT1dqpos\endcsname{127}
1523 \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

```

1524 \ifx\f@encoding\undefined
1525   \def\f@encoding{OT1}
1526 \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.

```

1527 \bbl@trace{Language attributes}
1528 \newcommand\languageattribute[2]{%
1529   \def\bbl@tempc{#1}%
1530   \bbl@fixname\bbl@tempc
1531   \bbl@iflanguage\bbl@tempc{%
1532     \bbl@vforeach{#2}{%

```

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

```

1533     \ifx\bbl@known@attribs\undefined
1534       \in@false
1535     \else
1536       \bbl@xin@{\,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1537     \fi
1538     \ifin@
1539       \bbl@warning{%
1540         You have more than once selected the attribute '##1'\%
1541         for language #1. Reported}%
1542     \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.

```

1543     \bbl@info{Activated '##1' attribute for\\%

```

```

1544         '\bbl@tempc'. Reported}%
1545     \bbl@exp{%
1546         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1547     \edef\bbl@tempa{\bbl@tempc-##1}%
1548     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1549     {\csname\bbl@tempc @attr##1\endcsname}%
1550     {\@attrerr{\bbl@tempc}{##1}}%
1551     \fi}}
1552 \@onlypreamble\languageattribute

```

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

```

1553 \newcommand*{\@attrerr}[2]{%
1554     \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}`.

```

1555 \def\bbl@declare@ttribute#1#2#3{%
1556     \bbl@xin@{,#2,}{,\BabelModifiers,}%
1557     \ifin@
1558         \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1559     \fi
1560     \bbl@add@list\bbl@attributes{#1-#2}%
1561     \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.

```

1562 \def\bbl@ifattributeset#1#2#3#4{%
1563     \ifx\bbl@known@attribs\@undefined
1564         \in@false
1565     \else
1566         \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1567     \fi
1568     \ifin@
1569         \bbl@afterelse#3%
1570     \else
1571         \bbl@afterfi#4%
1572     \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.

```

1573 \def\bbl@ifknown@ttrib#1#2{%
1574     \let\bbl@tempa\@secondoftwo
1575     \bbl@loopx\bbl@tempb{#2}{%
1576         \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1577     \ifin@
1578         \let\bbl@tempa\@firstoftwo
1579     \else
1580     \fi}%
1581     \bbl@tempa}

```


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

```

1582 \def\bbl@clear@ttribs{%
1583   \ifx\bbl@attributes\undefined\else
1584     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1585       \expandafter\bbl@clear@trib\bbl@tempa.}%
1586     \let\bbl@attributes\undefined
1587   \fi}
1588 \def\bbl@clear@trib#1-#2.{%
1589   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1590 \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.

```

1591 \bbl@trace{Macros for saving definitions}
1592 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1593 \newcount\babel@savecnt
1594 \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.

```

1595 \def\babel@save#1{%
1596   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1597   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1598     \expandafter{\expandafter,\bbl@savextras,}}%
1599   \expandafter\in@\bbl@tempa
1600   \ifin@else
1601     \bbl@add\bbl@savextras{,#1,}%
1602     \bbl@carg\let{\babel@number\babel@savecnt}#1\relax
1603     \toks@\expandafter{\originalTeX\let#1=}
1604     \bbl@exp{%
1605       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1606     \advance\babel@savecnt@ne
1607   \fi}
1608 \def\babel@savevariable#1{%
1609   \toks@\expandafter{\originalTeX #1=}
1610   \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 \LaTeX 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`.

```

1611 \def\bbl@redefine#1{%
1612   \edef\bbl@tempa{\bbl@stripslash#1}%
1613   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1614   \expandafter\def\csname\bbl@tempa\endcsname}
1615 \@onlypreamble\bbl@redefine

```

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

```

1616 \def\bbl@redefine@long#1{%
1617   \edef\bbl@tempa{\bbl@stripslash#1}%
1618   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1619   \long\expandafter\def\csname\bbl@tempa\endcsname}
1620 \@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`.

```

1621 \def\bbl@redefineroobust#1{%
1622   \edef\bbl@tempa{\bbl@stripslash#1}%
1623   \bbl@ifunset{\bbl@tempa\space}%
1624     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1625      \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1626     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1627     \@namedef{\bbl@tempa\space}}
1628 \@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.

```

1629 \def\bbl@frenchspacing{%
1630   \ifnum\the\sfcode`\.=\@m
1631     \let\bbl@nonfrenchspacing\relax
1632   \else
1633     \frenchspacing
1634     \let\bbl@nonfrenchspacing\nonfrenchspacing
1635   \fi}
1636 \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.

```

1637 \let\bbl@elt\relax
1638 \edef\bbl@fs@chars{%
1639   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1640   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1641   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1642 \def\bbl@pre@fs{%
1643   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1644   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1645 \def\bbl@post@fs{%
1646   \bbl@save@sfcodes
1647   \edef\bbl@tempa{\bbl@c{l}{frspc}}%
1648   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1649   \if u\bbl@tempa      % do nothing
1650   \else\if n\bbl@tempa % non french
1651     \def\bbl@elt##1##2##3{%
1652       \ifnum\sfcode`##1=##2\relax
1653       \babel@savevariable{\sfcode`##1}%

```

```

1654     \sfcode`##1=##3\relax
1655     \fi}%
1656 \bbl@fs@chars
1657 \else\if y\bbl@tempa % french
1658 \def\bbl@elt##1##2##3{%
1659     \ifnum\sfcode`##1=##3\relax
1660         \babel@savevariable{\sfcode`##1}%
1661         \sfcode`##1=##2\relax
1662     \fi}%
1663 \bbl@fs@chars
1664 \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.

```

1665 \bbl@trace{Hyphens}
1666 \@onlypreamble\babelhyphenation
1667 \AtEndOfPackage{%
1668     \newcommand\babelhyphenation[2][\@empty]{%
1669         \ifx\bbl@hyphenation@\relax
1670             \let\bbl@hyphenation@\@empty
1671         \fi
1672         \ifx\bbl@hyphlist@\@empty\else
1673             \bbl@warning{%
1674                 You must not intermingle \string\selectlanguage\space and\\%
1675                 \string\babelhyphenation\space or some exceptions will not\\%
1676                 be taken into account. Reported}%
1677         \fi
1678         \ifx\@empty#1%
1679             \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1680         \else
1681             \bbl@vforeach{#1}{%
1682                 \def\bbl@tempa{##1}%
1683                 \bbl@fixname\bbl@tempa
1684                 \bbl@iflanguage\bbl@tempa{%
1685                     \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1686                         \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1687                         }%
1688                     {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1689                     #2}}}%
1690         \fi}}

```

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

```

1691 \ifx\NewDocumentCommand\@undefined\else
1692     \NewDocumentCommand\babelhyphenmins{sommo}{%
1693         \IfNoValueTF{#2}%
1694             {\protected@edef\bbl@hyphenmins@\set@hyphenmins{#3}{#4}}%
1695             \IfValueT{#5}{%
1696                 \protected@edef\bbl@hyphenatmin@{\hyphenationmin=#5\relax}}%
1697             \IfBooleanT{#1}{%
1698                 \left@hyphenmin=#3\relax
1699                 \right@hyphenmin=#4\relax
1700                 \IfValueT{#5}{\hyphenationmin=#5\relax}}}%
1701     {\edef\bbl@tempb{\zap@space#2 \@empty}%
1702     \bbl@for\bbl@tempa\bbl@tempb{%
1703         \@namedef{bbl@hyphenmins@\bbl@tempa}{\set@hyphenmins{#3}{#4}}%
1704         \IfValueT{#5}{%
1705             \@namedef{bbl@hyphenatmin@\bbl@tempa}{\hyphenationmin=#5\relax}}}%
1706     \IfBooleanT{#1}{\bbl@error{hyphenmins-args}{#1}{#2}}}

```

1707 \fi

\bbl@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.

```
1708 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1709 \def\bbl@t@one{T1}
1710 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
```

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

```
1711 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1712 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1713 \def\bbl@hyphen{%
1714   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1715 \def\bbl@hyphen@i#1#2{%
1716   \lowercase{\bbl@ifunset{\bbl@hy@#1#2\@empty}}}%
1717   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1718   {\lowercase{\csname bbl@hy@#1#2\@empty\endcsname}}}
```

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

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

```
1719 \def\bbl@usehyphen#1{%
1720   \leavevmode
1721   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1722   \nobreak\hskip\z@skip}
1723 \def\bbl@@usehyphen#1{%
1724   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1725 \def\bbl@hyphenchar{%
1726   \ifnum\hyphenchar\font=\m@ne
1727     \babelnullhyphen
1728   \else
1729     \char\hyphenchar\font
1730   \fi}
```

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

```
1731 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1732 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1733 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1734 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1735 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1736 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1737 \def\bbl@hy@repeat{%
1738   \bbl@usehyphen{%
1739     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1740 \def\bbl@hy@@repeat{%
1741   \bbl@usehyphen{%
1742     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1743 \def\bbl@hy@empty{\hskip\z@skip}
1744 \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.

```
1745 \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.

```
1746 \bbl@trace{Multiencoding strings}
1747 \def\bbl@tglobal#1{\global\let#1#1}
```

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

```
1748 <<{*More package options}>> ≡
1749 \DeclareOption{nocase}{}
1750 <</More package options>>
```

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

```
1751 <<{*More package options}>> ≡
1752 \let\bbl@opt@strings\@nnil % accept strings=value
1753 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1754 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1755 \def\BabelStringsDefault{generic}
1756 <</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.

```
1757 \@onlypreamble\StartBabelCommands
1758 \def\StartBabelCommands{%
1759   \begingroup
1760   \@tempcnta="7F
1761   \def\bbl@tempa{%
1762     \ifnum\@tempcnta>"FF\else
1763       \catcode\@tempcnta=11
1764       \advance\@tempcnta\@ne
1765       \expandafter\bbl@tempa
1766     \fi}%
1767   \bbl@tempa
1768   <@Macros local to BabelCommands@>
1769   \def\bbl@provstring##1##2{%
1770     \providecommand##1{##2}%
1771     \bbl@tglobal##1}%
1772   \global\let\bbl@scafter\@empty
1773   \let\StartBabelCommands\bbl@startcmds
1774   \ifx\BabelLanguages\relax
1775     \let\BabelLanguages\CurrentOption
1776   \fi
1777   \begingroup
1778   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1779   \StartBabelCommands}
1780 \def\bbl@startcmds{%
1781   \ifx\bbl@screset\@nnil\else
1782     \bbl@usehooks{stopcommands}{}%
1783   \fi
1784   \endgroup
1785   \begingroup
1786   \@ifstar
1787     {\ifx\bbl@opt@strings\@nnil
1788       \let\bbl@opt@strings\BabelStringsDefault
1789     \fi
1790     \bbl@startcmds@i}%
1791   \bbl@startcmds@i}
1792 \def\bbl@startcmds@i#1#2{%
1793   \edef\bbl@L{\zap@space#1 \@empty}}%
```

```

1794 \edef\bbl@G{\zap@space#2 \@empty}%
1795 \bbl@startcmds@ii}
1796 \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.

```

1797 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1798 \let\SetString@gobbletwo
1799 \let\bbl@stringdef@gobbletwo
1800 \let\AfterBabelCommands@gobble
1801 \ifx\@empty#1%
1802 \def\bbl@sc@label{generic}%
1803 \def\bbl@encstring##1##2{%
1804 \ProvideTextCommandDefault##1{##2}%
1805 \bbl@tglobal##1%
1806 \expandafter\bbl@tglobal\csname\string?\string##1\endcsname}%
1807 \let\bbl@sctest\in@true
1808 \else
1809 \let\bbl@sc@charset\space % <- zapped below
1810 \let\bbl@sc@fontenc\space % <- " "
1811 \def\bbl@tempa##1=##2\@nil{%
1812 \bbl@csarg\edef{sc\zap@space##1 \@empty}{##2 }}%
1813 \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1814 \def\bbl@tempa##1 ##2{% space -> comma
1815 ##1%
1816 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1817 \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1818 \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1819 \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1820 \def\bbl@encstring##1##2{%
1821 \bbl@foreach\bbl@sc@fontenc{%
1822 \bbl@ifunset{T@####1}%
1823 }%
1824 {\ProvideTextCommand##1{####1}{##2}%
1825 \bbl@tglobal##1%
1826 \expandafter
1827 \bbl@tglobal\csname####1\string##1\endcsname}}}%
1828 \def\bbl@sctest{%
1829 \bbl@xin{\, \bbl@opt@strings,}{, \bbl@sc@label, \bbl@sc@fontenc,}%
1830 \fi
1831 \ifx\bbl@opt@strings\@nnil % i.e., no strings key -> defaults
1832 \else\ifx\bbl@opt@strings\relax % i.e., strings=encoded
1833 \let\AfterBabelCommands\bbl@aftercmds
1834 \let\SetString\bbl@setstring
1835 \let\bbl@stringdef\bbl@encstring
1836 \else % i.e., strings=value
1837 \bbl@sctest
1838 \fin@
1839 \let\AfterBabelCommands\bbl@aftercmds
1840 \let\SetString\bbl@setstring
1841 \let\bbl@stringdef\bbl@provstring
1842 \fi\fi\fi
1843 \bbl@scswitch
1844 \ifx\bbl@G\@empty
1845 \def\SetString##1##2{%
1846 \bbl@error{missing-group}{##1}{}}%

```

```

1847 \fi
1848 \ifx\@empty#1%
1849 \bbl@usehooks{defaultcommands}{}%
1850 \else
1851 \@expandtwoargs
1852 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1853 \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).

```

1854 \def\bbl@forlang#1#2{%
1855 \bbl@for#1\bbl@L{%
1856 \bbl@xin@{,#1,}{,\BabelLanguages,}%
1857 \ifin#2\relax\fi}}
1858 \def\bbl@scswitch{%
1859 \bbl@forlang\bbl@tempa{%
1860 \ifx\bbl@G\@empty\else
1861 \ifx\SetString@gobbletwo\else
1862 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1863 \bbl@xin@{\bbl@GL,}{,\bbl@screset,}%
1864 \ifin\else
1865 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1866 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1867 \fi
1868 \fi
1869 \fi}}
1870 \AtEndOfPackage{%
1871 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1872 \let\bbl@scswitch\relax}
1873 \@onlypreamble\EndBabelCommands
1874 \def\EndBabelCommands{%
1875 \bbl@usehooks{stopcommands}{}%
1876 \endgroup
1877 \endgroup
1878 \bbl@scafter}
1879 \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.

```

1880 \def\bbl@setstring#1#2{ e.g., \prefacename<string>}
1881 \bbl@forlang\bbl@tempa{%
1882 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1883 \bbl@ifunset{\bbl@LC} e.g., \germanchaptername
1884 {\bbl@exp{%
1885 \global\\bbl@add\<\bbl@G\bbl@tempa>{\\bbl@scset\\#1\<\bbl@LC>}}}%
1886 }%
1887 \def\BabelString{#2}%
1888 \bbl@usehooks{stringprocess}{}%
1889 \expandafter\bbl@stringdef
1890 \csname\bbl@LC\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`.

```
1891 \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.

```
1892 <<{*Macros local to BabelCommands}>> ≡
1893 \def\SetStringLoop##1##2{%
1894   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1895   \count@\z@
1896   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1897     \advance\count@\@ne
1898     \toks@\expandafter{\bbl@tempa}%
1899     \bbl@exp{%
1900       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1901       \count@=\the\count@relax}}}%
1902 <</Macros local to BabelCommands>>
```

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

```
1903 \def\bbl@aftercmds#1{%
1904   \toks@\expandafter{\bbl@scafter#1}%
1905   \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.

```
1906 <<{*Macros local to BabelCommands}>> ≡
1907 \newcommand\SetCase[3][]{%
1908   \def\bbl@tempa####1####2{%
1909     \ifx####1\@empty\else
1910       \bbl@carg\bbl@add{extras\CurrentOption}{%
1911         \bbl@carg\babel@save{c__text_uppercase_\string####1_tl}%
1912         \bbl@carg\def{c__text_uppercase_\string####1_tl}{####2}%
1913         \bbl@carg\babel@save{c__text_lowercase_\string####2_tl}%
1914         \bbl@carg\def{c__text_lowercase_\string####2_tl}{####1}}%
1915       \expandafter\bbl@tempa
1916       \fi}%
1917   \bbl@tempa##1\@empty\@empty
1918   \bbl@carg\bbl@tglobal{extras\CurrentOption}}%
1919 <</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.

```
1920 <<{*Macros local to BabelCommands}>> ≡
1921 \newcommand\SetHyphenMap[1]{%
1922   \bbl@forlang\bbl@tempa{%
1923     \expandafter\bbl@stringdef
1924     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1925 <</Macros local to BabelCommands>>
```

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

```
1926 \newcommand\BabelLower[2]{% one to one.
1927   \ifnum\lccode#1=#2\else
1928     \babel@savevariable{\lccode#1}%
1929     \lccode#1=#2\relax
1930   \fi}
1931 \newcommand\BabelLowerMM[4]{% many-to-many
1932   \@tempcnta=#1\relax
1933   \@tempcntb=#4\relax
1934   \def\bbl@tempa{%
1935     \ifnum\@tempcnta>#2\else
1936       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1937       \advance\@tempcnta#3\relax
```



```

1938     \advance\@tempcntb#3\relax
1939     \expandafter\bbbl@tempa
1940     \fi}%
1941     \bbbl@tempa}
1942 \newcommand\BabelLowerM0[4]{% many-to-one
1943     \@tempcnta=#1\relax
1944     \def\bbbl@tempa{%
1945         \ifnum\@tempcnta>#2\else
1946             \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1947             \advance\@tempcnta#3
1948             \expandafter\bbbl@tempa
1949             \fi}%
1950     \bbbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1951 <<{*More package options}>> ≡
1952 \DeclareOption{hyphenmap=off}{\chardef\bbbl@opt@hyphenmap\z@}
1953 \DeclareOption{hyphenmap=first}{\chardef\bbbl@opt@hyphenmap\@ne}
1954 \DeclareOption{hyphenmap=select}{\chardef\bbbl@opt@hyphenmap\tw@}
1955 \DeclareOption{hyphenmap=other}{\chardef\bbbl@opt@hyphenmap\thr@@}
1956 \DeclareOption{hyphenmap=other*}{\chardef\bbbl@opt@hyphenmap4\relax}
1957 <</More package options>>

```

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

```

1958 \AtEndOfPackage{%
1959     \ifx\bbbl@opt@hyphenmap\undefined
1960         \bbbl@xin@{,}{\bbbl@language@opts}%
1961         \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
1962     \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.

```

1963 \newcommand\setlocalecaption{%
1964     \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
1965 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
1966     \bbbl@trim@def\bbbl@tempa{#2}%
1967     \bbbl@xin@{.template}{\bbbl@tempa}%
1968     \ifin@
1969         \bbbl@ini@captions@template{#3}{#1}%
1970     \else
1971         \edef\bbbl@tempd{%
1972             \expandafter\expandafter\expandafter
1973             \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1974         \bbbl@xin@
1975             {\expandafter\string\csname #2name\endcsname}%
1976             {\bbbl@tempd}%
1977         \ifin@ % Renew caption
1978             \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
1979             \ifin@
1980                 \bbbl@exp{%
1981                     \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1982                     {\bbbl@scset\<#2name>\<#1#2name>}}%
1983                 {}}%
1984         \else % Old way converts to new way
1985             \bbbl@ifunset{#1#2name}%
1986             {\bbbl@exp{%
1987                 \\bbbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
1988                 \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
1989                 {\def\<#2name>{\<#1#2name>}}%
1990                 {}}}%
1991             {}%

```

```

1992     \fi
1993 \else
1994     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1995     \ifin@ % New way
1996     \bbl@exp{%
1997         \\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
1998         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
1999         {\\\bbl@scset\<#2name>\<#1#2name>}%
2000         }%
2001     \else % Old way, but defined in the new way
2002     \bbl@exp{%
2003         \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2004         \\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2005         {\def\<#2name>{\<#1#2name>}}%
2006         }%
2007     \fi%
2008 \fi
2009 \@namedef{#1#2name}{#3}%
2010 \toks@ \expandafter{\bbl@captionslist}%
2011 \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2012 \ifin\else
2013     \bbl@exp{\\\bbl@add\\bbl@captionslist{\<#2name>}}%
2014     \bbl@tglobal\bbl@captionslist
2015 \fi
2016 \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.

```

2017 \bbl@trace{Macros related to glyphs}
2018 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2019     \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2020     \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.

```

2021 \def\save@sf@q#1{\leavevmode
2022     \begingroup
2023     \edef\@SF{\spacefactor\the\spacefactor}\@SF
2024     \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.

```

2025 \ProvideTextCommand{\quotedblbase}{OT1}{%
2026     \save@sf@q{\set@low@box{\textquotedblright}/}%
2027     \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.

```

2028 \ProvideTextCommandDefault{\quotedblbase}{%
2029     \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2030 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2031     \save@sf@q{\set@low@box{\textquoteright}/}%
2032     \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.

```
2033 \ProvideTextCommandDefault{\quotesinglbase}{%
2034 \UseTextSymbol{OT1}{\quotesinglbase}}
```

\guillemetleft

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

```
2035 \ProvideTextCommand{\guillemetleft}{OT1}{%
2036 \ifmmode
2037 \ll
2038 \else
2039 \save@sf@q{\nobreak
2040 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2041 \fi}
2042 \ProvideTextCommand{\guillemetright}{OT1}{%
2043 \ifmmode
2044 \gg
2045 \else
2046 \save@sf@q{\nobreak
2047 \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
2048 \fi}
2049 \ProvideTextCommand{\guillemotleft}{OT1}{%
2050 \ifmmode
2051 \ll
2052 \else
2053 \save@sf@q{\nobreak
2054 \raise.2ex\hbox{\scriptscriptstyle\ll}\bbl@allowhyphens}%
2055 \fi}
2056 \ProvideTextCommand{\guillemotright}{OT1}{%
2057 \ifmmode
2058 \gg
2059 \else
2060 \save@sf@q{\nobreak
2061 \raise.2ex\hbox{\scriptscriptstyle\gg}\bbl@allowhyphens}%
2062 \fi}
```

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

```
2063 \ProvideTextCommandDefault{\guillemetleft}{%
2064 \UseTextSymbol{OT1}{\guillemetleft}}
2065 \ProvideTextCommandDefault{\guillemetright}{%
2066 \UseTextSymbol{OT1}{\guillemetright}}
2067 \ProvideTextCommandDefault{\guillemotleft}{%
2068 \UseTextSymbol{OT1}{\guillemotleft}}
2069 \ProvideTextCommandDefault{\guillemotright}{%
2070 \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft

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

```
2071 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2072 \ifmmode
2073 <%
2074 \else
2075 \save@sf@q{\nobreak
2076 \raise.2ex\hbox{\scriptscriptstyle<}\bbl@allowhyphens}%
2077 \fi}
2078 \ProvideTextCommand{\guilsinglright}{OT1}{%
2079 \ifmmode
2080 >%
2081 \else
2082 \save@sf@q{\nobreak
2083 \raise.2ex\hbox{\scriptscriptstyle>}\bbl@allowhyphens}%
2084 \fi}
```

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

```
2085 \ProvideTextCommandDefault{\guilsinglleft}{%
2086 \UseTextSymbol{OT1}{\guilsinglleft}}
2087 \ProvideTextCommandDefault{\guilsinglright}{%
2088 \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.

```
2089 \DeclareTextCommand{\ij}{OT1}{%
2090 i\kern-0.02em\bbl@allowhyphens j}
2091 \DeclareTextCommand{\IJ}{OT1}{%
2092 I\kern-0.02em\bbl@allowhyphens J}
2093 \DeclareTextCommand{\ij}{T1}{\char188}
2094 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2095 \ProvideTextCommandDefault{\ij}{%
2096 \UseTextSymbol{OT1}{\ij}}
2097 \ProvideTextCommandDefault{\IJ}{%
2098 \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).

```
2099 \def\crrtic@{\hrule height0.1ex width0.3em}
2100 \def\crrtic@{\hrule height0.1ex width0.33em}
2101 \def\ddj@{%
2102 \setbox0\hbox{d}\dimen@=\ht0
2103 \advance\dimen@lex
2104 \dimen@.45\dimen@
2105 \dimen@ii\expandafter\rem@pt\the\fontdimen@ne\font\dimen@
2106 \advance\dimen@ii.5ex
2107 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2108 \def\DDJ@{%
2109 \setbox0\hbox{D}\dimen@=.55\ht0
2110 \dimen@ii\expandafter\rem@pt\the\fontdimen@ne\font\dimen@
2111 \advance\dimen@ii.15ex % correction for the dash position
2112 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2113 \dimen\thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2114 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2115 %
2116 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2117 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2118 \ProvideTextCommandDefault{\dj}{%
2119 \UseTextSymbol{OT1}{\dj}}
2120 \ProvideTextCommandDefault{\DJ}{%
2121 \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.

```
2122 \DeclareTextCommand{\SS}{OT1}{SS}
2123 \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.

```
2124 \ProvideTextCommandDefault{\glq}{%
2125 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2126 \ProvideTextCommand{\grq}{T1}{%
2127 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2128 \ProvideTextCommand{\grq}{TU}{%
2129 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2130 \ProvideTextCommand{\grq}{OT1}{%
2131 \save@sf@q{\kern-.0125em
2132 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2133 \kern.07em\relax}}
2134 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq`

`\grqq` The ‘german’ double quotes.

```
2135 \ProvideTextCommandDefault{\glqq}{%
2136 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2137 \ProvideTextCommand{\grqq}{T1}{%
2138 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}}
2139 \ProvideTextCommand{\grqq}{TU}{%
2140 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}}
2141 \ProvideTextCommand{\grqq}{OT1}{%
2142 \save@sf@q{\kern-.07em
2143 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2144 \kern.07em\relax}}
2145 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq`

`\frq` The ‘french’ single guillemets.

```
2146 \ProvideTextCommandDefault{\flq}{%
2147 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}}
2148 \ProvideTextCommandDefault{\frq}{%
2149 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}}
```

`\flqq`

`\frqq` The ‘french’ double guillemets.

```
2150 \ProvideTextCommandDefault{\flqq}{%
2151 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}}
2152 \ProvideTextCommandDefault{\frqq}{%
2153 \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).

```

2154 \def\umlauthigh{%
2155   \def\bbbl@umlauta##1{\leavevmode\bgroup%
2156     \accent\csname\f@encoding dqpos\endcsname
2157     ##1\bbbl@allowhyphens\egroup}%
2158   \let\bbbl@umlaute\bbbl@umlauta}
2159 \def\umlautlow{%
2160   \def\bbbl@umlauta{\protect\lower@umlaut}}
2161 \def\umlautelow{%
2162   \def\bbbl@umlaute{\protect\lower@umlaut}}
2163 \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.

```

2164 \expandafter\ifx\csname U@D\endcsname\relax
2165   \csname newdimen\endcsname\U@D
2166 \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.

```

2167 \def\lower@umlaut#1{%
2168   \leavevmode\bgroup
2169   \U@D lex%
2170   {\setbox\z@\hbox{%
2171     \char\csname\f@encoding dqpos\endcsname}%
2172     \dimen@ -.45ex\advance\dimen@\ht\z@
2173     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2174   \accent\csname\f@encoding dqpos\endcsname
2175   \fontdimen5\font\U@D #1%
2176   \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bbbl@umlauta` or `\bbbl@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 `\bbbl@umlauta` and/or `\bbbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2177 \AtBeginDocument{%
2178   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbbl@umlauta{a}}%
2179   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbbl@umlaute{e}}%
2180   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbbl@umlaute{i}}%
2181   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbbl@umlaute{i}}%
2182   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbbl@umlauta{o}}%
2183   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbbl@umlauta{u}}%
2184   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbbl@umlauta{A}}%
2185   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbbl@umlaute{E}}%
2186   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbbl@umlaute{I}}%
2187   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbbl@umlauta{O}}%
2188   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbbl@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`.

```

2189 \ifx\l@english\undefined
2190   \chardef\l@english\z@
2191 \fi

```

```

2192 % The following is used to cancel rules in ini files (see Amharic).
2193 \ifx\l@unhyphenated\@undefined
2194 \newlanguage\l@unhyphenated
2195 \fi

```

4.16. Layout

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

```

2196 \bbl@trace{Bidi layout}
2197 \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.

```

2198 \bbl@trace{Input engine specific macros}
2199 \ifcase\bbl@engine
2200 \input txtbabel.def
2201 \or
2202 \input luababel.def
2203 \or
2204 \input xebabel.def
2205 \fi
2206 \providecommand\babelfont{\bbl@error{only-lua-xe}{}{}}
2207 \providecommand\babelprehyphenation{\bbl@error{only-lua}{}{}}
2208 \ifx\babelposthyphenation\@undefined
2209 \let\babelposthyphenation\babelprehyphenation
2210 \let\babelpatterns\babelprehyphenation
2211 \let\babelcharproperty\babelprehyphenation
2212 \fi
2213 </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.

```

2214 <{*package}>
2215 \bbl@trace{Creating languages and reading ini files}
2216 \let\bbl@extend@ini@gobble
2217 \newcommand\babelprovide[2][]{%
2218 \let\bbl@savelangname\languagename
2219 \edef\bbl@savelocaleid{\the\localeid}%
2220 % Set name and locale id
2221 \edef\languagename{#2}%
2222 \bbl@id@assign
2223 % Initialize keys
2224 \bbl@vforeach{captions,date,import,main,script,language,%
2225 hyphenrules,linebreaking,justification,mapfont,maparabic,%
2226 mapdigits,intraspaces,intrapenalty,onchar,transforms,alph,%
2227 Alph,labels,labels*,mapdot,calendar,date,casing,interchar,%
2228 @import}%
2229 {\bbl@csarg\let{KVP@##1}\@nnil}%
2230 \global\let\bbl@release@transforms@empty
2231 \global\let\bbl@release@casing@empty
2232 \let\bbl@calendars@empty
2233 \global\let\bbl@inidata@empty
2234 \global\let\bbl@extend@ini@gobble
2235 \global\let\bbl@included@inis@empty
2236 \gdef\bbl@key@list{;}%
2237 \bbl@ifunset\bbl@passto@#2}%

```

```

2238     {\def\bbl@tempa{#1}}%
2239     {\bbl@exp{\def\\bbl@tempa{[bbl@passto@#2],\unexpanded{#1}}}%
2240 \expandafter\bbl@forkv\expandafter{\bbl@tempa}{%
2241 \in@{/}{##1}% With /, (re)sets a value in the ini
2242 \ifin@
2243 \bbl@renewinikey##1\@{##2}%
2244 \else
2245 \bbl@csarg\ifx{KVP@##1}\@nnil\else
2246 \bbl@error{unknown-provide-key}{##1}{}}%
2247 \fi
2248 \bbl@csarg\def{KVP@##1}{##2}%
2249 \fi}%
2250 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2251 \bbl@ifunset{date#2}\z{\bbl@ifunset{bbl@llevel@#2}\@ne\tw}%
2252 % == init ==
2253 \ifx\bbl@screset\@undefined
2254 \bbl@ldfinit
2255 \fi
2256 % ==
2257 % If there is no import (last wins), use @import (internal, there
2258 % must be just one). To consider any order (because
2259 % \PassOptionsToLocale).
2260 \ifx\bbl@KVP@import\@nnil
2261 \let\bbl@KVP@import\bbl@KVP@import
2262 \fi
2263 % == date (as option) ==
2264 % \ifx\bbl@KVP@date\@nnil\else
2265 % \fi
2266 % ==
2267 \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2268 \ifcase\bbl@howloaded
2269 \let\bbl@lbkflag\@empty % new
2270 \else
2271 \ifx\bbl@KVP@hyphenrules\@nnil\else
2272 \let\bbl@lbkflag\@empty
2273 \fi
2274 \ifx\bbl@KVP@import\@nnil\else
2275 \let\bbl@lbkflag\@empty
2276 \fi
2277 \fi
2278 % == import, captions ==
2279 \ifx\bbl@KVP@import\@nnil\else
2280 \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2281 {\ifx\bbl@initload\relax
2282 \begingroup
2283 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2284 \bbl@input@texini{#2}%
2285 \endgroup
2286 \else
2287 \xdef\bbl@KVP@import{\bbl@initload}%
2288 \fi}%
2289 }%
2290 \let\bbl@KVP@date\@empty
2291 \fi
2292 \let\bbl@KVP@captions@\bbl@KVP@captions
2293 \ifx\bbl@KVP@captions\@nnil
2294 \let\bbl@KVP@captions\bbl@KVP@import
2295 \fi
2296 % ==
2297 \ifx\bbl@KVP@transforms\@nnil\else
2298 \bbl@replace\bbl@KVP@transforms{ },}%
2299 \fi
2300 % ==

```



```

2301 \ifx\bbk@KVP@mapdot\@nnil\else
2302 \def\bbk@tempa{\@empty}%
2303 \ifx\bbk@KVP@mapdot\bbk@tempa\else
2304 \bbk@exp{\gdef\<bbk@map@. @\language>{\bbk@KVP@mapdot}}}%
2305 \fi
2306 \fi
2307 % Load ini
2308 % -----
2309 \ifcase\bbk@howloaded
2310 \bbk@provide@new{#2}%
2311 \else
2312 \bbk@ifblank{#1}%
2313 {}% With \bbk@load@basic below
2314 {\bbk@provide@renew{#2}}}%
2315 \fi
2316 % Post tasks
2317 % -----
2318 % == subsequent calls after the first provide for a locale ==
2319 \ifx\bbk@inidata\@empty\else
2320 \bbk@extend@ini{#2}%
2321 \fi
2322 % == ensure captions ==
2323 \ifx\bbk@KVP@captions\@nnil\else
2324 \bbk@ifunset{bbk@extracaps@#2}%
2325 {\bbk@exp{\babelensure[exclude=\\today]{#2}}}%
2326 {\bbk@exp{\babelensure[exclude=\\today,
2327 include=\bbk@extracaps@#2]}{#2}}}%
2328 \bbk@ifunset{bbk@ensure@\language}%
2329 {\bbk@exp{%
2330 \\DeclareRobustCommand\<bbk@ensure@\language>[1]{%
2331 \\foreignlanguage{\language}%
2332 {###1}}}%
2333 }%
2334 \bbk@exp{%
2335 \\bbk@tglobal\<bbk@ensure@\language>%
2336 \\bbk@tglobal\<bbk@ensure@\language\space>}%
2337 \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.

```

2338 \bbk@load@basic{#2}%
2339 % == script, language ==
2340 % Override the values from ini or defines them
2341 \ifx\bbk@KVP@script\@nnil\else
2342 \bbk@csarg\edef{sname@#2}{\bbk@KVP@script}%
2343 \fi
2344 \ifx\bbk@KVP@language\@nnil\else
2345 \bbk@csarg\edef{lname@#2}{\bbk@KVP@language}%
2346 \fi
2347 \ifcase\bbk@engine\or
2348 \bbk@ifunset{bbk@chrng@\language}{}%
2349 {\directlua{
2350 Babel.set_chranges_b('\bbk@cl{sbc}', '\bbk@cl{chrng}') }}%
2351 \fi
2352 % == Line breaking: intraspace, intrapenalty ==
2353 % For CJK, East Asian, Southeast Asian, if interspace in ini
2354 \ifx\bbk@KVP@intraspace\@nnil\else % We can override the ini or set
2355 \bbk@csarg\edef{intsp@#2}{\bbk@KVP@intraspace}%
2356 \fi
2357 \bbk@provide@intraspace
2358 % == Line breaking: justification ==
2359 \ifx\bbk@KVP@justification\@nnil\else

```

```

2360 \let\bbl@KVP@linebreaking\bbl@KVP@justification
2361 \fi
2362 \ifx\bbl@KVP@linebreaking\@nnil\else
2363 \bbl@xin@{\,\bbl@KVP@linebreaking,}%
2364 {,elongated,kashida,cjk,padding,unhyphenated,}%
2365 \ifin@
2366 \bbl@csarg\xdef
2367 {\lbrk@language}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2368 \fi
2369 \fi
2370 \bbl@xin@{/e}{/\bbl@cl{lbrk}}%
2371 \ifin@else\bbl@xin@{/k}{/\bbl@cl{lbrk}}\fi
2372 \ifin@\bbl@arabicjust\fi
2373 \bbl@xin@{/p}{/\bbl@cl{lbrk}}%
2374 \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2375 % == Line breaking: hyphenate.other.(locale|script) ==
2376 \ifx\bbl@lbrkflag\@empty
2377 \bbl@ifunset{bbl@hyotl@language}{}%
2378 {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
2379 \bbl@startcommands*{language}{}%
2380 \bbl@csarg\bbl@foreach{hyotl@language}{%
2381 \ifcase\bbl@engine
2382 \ifnum##1<257
2383 \SetHyphenMap{\BabelLower{##1}{##1}}%
2384 \fi
2385 \else
2386 \SetHyphenMap{\BabelLower{##1}{##1}}%
2387 \fi}%
2388 \bbl@endcommands}%
2389 \bbl@ifunset{bbl@hyots@language}{}%
2390 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
2391 \bbl@csarg\bbl@foreach{hyots@language}{%
2392 \ifcase\bbl@engine
2393 \ifnum##1<257
2394 \global\lccode##1=##1\relax
2395 \fi
2396 \else
2397 \global\lccode##1=##1\relax
2398 \fi}}%
2399 \fi
2400 % == Counters: maparabic ==
2401 % Native digits, if provided in ini (TeX level, xe and lua)
2402 \ifcase\bbl@engine\else
2403 \bbl@ifunset{bbl@dgnat@language}{}%
2404 {\expandafter\ifx\cname bbl@dgnat@language\endcsname\@empty\else
2405 \expandafter\expandafter\expandafter
2406 \bbl@setdigits\cname bbl@dgnat@language\endcsname
2407 \ifx\bbl@KVP@maparabic\@nnil\else
2408 \ifx\bbl@latinarabic\@undefined
2409 \expandafter\let\expandafter\@arabic
2410 \cname bbl@counter@language\endcsname
2411 \else % i.e., if layout=counters, which redefines \@arabic
2412 \expandafter\let\expandafter\bbl@latinarabic
2413 \cname bbl@counter@language\endcsname
2414 \fi
2415 \fi
2416 \fi}%
2417 \fi
2418 % == Counters: mapdigits ==
2419 % > luababel.def
2420 % == Counters: alph, Alph ==
2421 \ifx\bbl@KVP@alph\@nnil\else
2422 \bbl@exp{%

```

```

2423     \\bbl@add\<bbl@preextras@\languagename>{%
2424     \\babel@save\\ \@alph
2425     \let\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2426 \fi
2427 \ifx\bbl@KVP@Alph\@nnil\else
2428     \bbl@exp{%
2429     \\bbl@add\<bbl@preextras@\languagename>{%
2430     \\babel@save\\ \@Alph
2431     \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2432 \fi
2433 % == Counters: mapdot ==
2434 \ifx\bbl@KVP@mapdot\@nnil\else
2435     \bbl@foreach\bbl@list@the{%
2436     \bbl@ifunset{the##1}}{%
2437     {{\bbl@ncarg\let\bbl@tempd{the##1}%
2438     \bbl@carg\bbl@sreplace{the##1}{.}{\bbl@map@lbl{.}}%
2439     \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
2440     \bbl@exp{\gdef\<the##1>{\[the##1]}}%
2441     \fi}}%
2442     \edef\bbl@tempb{enumi,enumii,enumiii,enumiv}%
2443     \bbl@foreach\bbl@tempb{%
2444     \bbl@ifunset{label##1}}{%
2445     {{\bbl@ncarg\let\bbl@tempd{label##1}%
2446     \bbl@carg\bbl@sreplace{label##1}{.}{\bbl@map@lbl{.}}%
2447     \expandafter\ifx\csname label##1\endcsname\bbl@tempd\else
2448     \bbl@exp{\gdef\<label##1>{\[label##1]}}%
2449     \fi}}%
2450 \fi
2451 % == Casing ==
2452 \bbl@release@casing
2453 \ifx\bbl@KVP@casing\@nnil\else
2454     \bbl@csarg\xdef{casing@\languagename}%
2455     {\@nameuse{bbl@casing@\languagename}\bbl@maybextx\bbl@KVP@casing}%
2456 \fi
2457 % == Calendars ==
2458 \ifx\bbl@KVP@calendar\@nnil
2459     \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2460 \fi
2461 \def\bbl@tempe##1 ##2\@{% Get first calendar
2462     \def\bbl@tempa{##1}}%
2463     \bbl@exp{\@bbl@tempe\bbl@KVP@calendar\space\\ \@}%
2464 \def\bbl@tempe##1.##2.##3\@{%
2465     \def\bbl@tempc{##1}%
2466     \def\bbl@tempb{##2}}%
2467 \expandafter\bbl@tempe\bbl@tempa.. \@
2468 \bbl@csarg\edef{calpr@\languagename}{%
2469     \ifx\bbl@tempc\@empty\else
2470     calendar=\bbl@tempc
2471     \fi
2472     \ifx\bbl@tempb\@empty\else
2473     ,variant=\bbl@tempb
2474     \fi}%
2475 % == engine specific extensions ==
2476 % Defined in XXXbabel.def
2477 \bbl@provide@extra{#2}%
2478 % == require.babel in ini ==
2479 % To load or reload the babel-*.tex, if require.babel in ini
2480 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2481     \bbl@ifunset{bbl@rqtex@\languagename}}{%
2482     {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2483     \let\BabelBeforeIni@gobbletwo
2484     \chardef\atcatcode=\catcode`\@
2485     \catcode`\@=11\relax

```

```

2486     \def\CurrentOption{#2}%
2487     \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2488     \catcode`\@=\atcatcode
2489     \let\atcatcode\relax
2490     \global\bbl@csarg\let{rqtex@\language}\relax
2491     \fi}%
2492 \bbl@foreach\bbl@calendars{%
2493   \bbl@ifunset{bbl@ca##1}{%
2494     \chardef\atcatcode=\catcode`@
2495     \catcode`\@=11\relax
2496     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2497     \catcode`\@=\atcatcode
2498     \let\atcatcode\relax}%
2499   {}}%
2500 \fi
2501 % == frenchspacing ==
2502 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2503 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2504 \ifin@
2505   \bbl@extras@wrap{\bbl@pre@fs}%
2506   {\bbl@pre@fs}%
2507   {\bbl@post@fs}%
2508 \fi
2509 % == transforms ==
2510 % > luababel.def
2511 \def\CurrentOption{#2}%
2512 \@nameuse{bbl@icsave@#2}%
2513 % == main ==
2514 \ifx\bbl@KVP@main@nnil % Restore only if not 'main'
2515   \let\language\bbl@savelangname
2516   \chardef\localeid\bbl@savelocaleid\relax
2517 \fi
2518 % == hyphenrules (apply if current) ==
2519 \ifx\bbl@KVP@hyphenrules@nnil\else
2520   \ifnum\bbl@savelocaleid=\localeid
2521     \language\@nameuse{l@\language}%
2522   \fi
2523 \fi}

```

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

```

2524 \def\bbl@provide@new#1{%
2525   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2526   \@namedef{extras#1}{}%
2527   \@namedef{noextras#1}{}%
2528   \bbl@startcommands*{#1}{captions}%
2529   \ifx\bbl@KVP@captions@nnil % and also if import, implicit
2530     \def\bbl@tempb##1{} % elt for \bbl@captionslist
2531     \ifx##1@nnil\else
2532       \bbl@exp{%
2533         \\SetString\\##1%
2534         \\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}%
2535       \expandafter\bbl@tempb
2536     \fi}%
2537   \expandafter\bbl@tempb\bbl@captionslist@nnil
2538 \else
2539   \ifx\bbl@initoload\relax
2540     \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2541   \else
2542     \bbl@read@ini{\bbl@initoload}2% % Same
2543   \fi
2544 \fi
2545 \StartBabelCommands*{#1}{date}%

```

```

2546 \ifx\bbk@KVP@date\@nnil
2547 \bbk@exp{%
2548 \SetString\@today{\@bbk@nocaption{today}{#1today}}}%
2549 \else
2550 \bbk@savetoday
2551 \bbk@savedate
2552 \fi
2553 \bbk@endcommands
2554 \bbk@load@basic{#1}%
2555 % == hyphenmins == (only if new)
2556 \bbk@exp{%
2557 \gdef\<#1hyphenmins>{%
2558 {\bbk@ifunset{bbk@lfthm@#1}{2}{\bbk@cs{lfthm@#1}}}%
2559 {\bbk@ifunset{bbk@rgthm@#1}{3}{\bbk@cs{rgthm@#1}}}}}%
2560 % == hyphenrules (also in renew) ==
2561 \bbk@provide@hyphens{#1}%
2562 % == main ==
2563 \ifx\bbk@KVP@main\@nnil\else
2564 \expandafter\main@language\expandafter{#1}%
2565 \fi}
2566%
2567 \def\bbk@provide@renew#1{%
2568 \ifx\bbk@KVP@captions\@nnil\else
2569 \StartBabelCommands*{#1}{captions}%
2570 \bbk@read@ini{\bbk@KVP@captions}2% % Here all letters cat = 11
2571 \EndBabelCommands
2572 \fi
2573 \ifx\bbk@KVP@date\@nnil\else
2574 \StartBabelCommands*{#1}{date}%
2575 \bbk@savetoday
2576 \bbk@savedate
2577 \EndBabelCommands
2578 \fi
2579 % == hyphenrules (also in new) ==
2580 \ifx\bbk@lbfkflag\@empty
2581 \bbk@provide@hyphens{#1}%
2582 \fi
2583 % == main ==
2584 \ifx\bbk@KVP@main\@nnil\else
2585 \expandafter\main@language\expandafter{#1}%
2586 \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.

```

2587 \def\bbk@load@basic#1{%
2588 \ifcase\bbk@howloaded\or\or
2589 \ifcase\csname bbl@llevel@\language\endcsname
2590 \bbk@csarg\let{lname@\language}\relax
2591 \fi
2592 \fi
2593 \bbk@ifunset{bbk@lname@#1}%
2594 {\def\BabelBeforeIni##1##2{%
2595 \begingroup
2596 \let\bbk@ini@captions@aux\@gobbletwo
2597 \def\bbk@inidate ####1.####2.####3.####4\relax ####5####6}%
2598 \bbk@read@ini{##1}1%
2599 \ifx\bbk@initoload\relax\endinput\fi
2600 \endgroup}%
2601 \begingroup % boxed, to avoid extra spaces:
2602 \ifx\bbk@initoload\relax
2603 \bbk@input@texini{#1}%
2604 \else

```

```

2605     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2606     \fi
2607     \endgroup}%
2608     {}

```

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).

```

2609 \def\bbl@load@info#1{%
2610   \def\BabelBeforeIni##1##2{%
2611     \begingroup
2612     \bbl@read@ini{##1}0%
2613     \endinput           % babel- .tex may contain onlypreamble's
2614     \endgroup}%       boxed, to avoid extra spaces:
2615     {\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.

```

2616 \def\bbl@provide@hyphens#1{%
2617   \@tempcnta\m@ne % a flag
2618   \ifx\bbl@KVP@hyphenrules\@nnil\else
2619     \bbl@replace\bbl@KVP@hyphenrules{ },}%
2620     \bbl@foreach\bbl@KVP@hyphenrules{%
2621       \ifnum\@tempcnta=\m@ne % if not yet found
2622         \bbl@ifsamestring{##1}{+}%
2623         {\bbl@carg\addlanguage{l@##1}}%
2624         {}%
2625         \bbl@ifunset{l@##1}% After a possible +
2626         {}%
2627         {\@tempcnta\@nameuse{l@##1}}%
2628         \fi}%
2629     \ifnum\@tempcnta=\m@ne
2630       \bbl@warning{%
2631         Requested 'hyphenrules' for '\language' not found:\%
2632         \bbl@KVP@hyphenrules.\%
2633         Using the default value. Reported}%
2634       \fi
2635     \fi
2636     \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2637       \ifx\bbl@KVP@captions@\@nnil
2638         \bbl@ifunset{\bbl@hyphr@#1}{}% use value in ini, if exists
2639         {\bbl@exp{\@bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2640          {}%
2641          {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2642          {}% if hyphenrules found:
2643          {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2644       \fi
2645     \fi
2646     \bbl@ifunset{l@#1}%
2647     {\ifnum\@tempcnta=\m@ne
2648       \bbl@carg\adddialect{l@#1}\language
2649       \else
2650       \bbl@carg\adddialect{l@#1}\@tempcnta
2651       \fi}%
2652     {\ifnum\@tempcnta=\m@ne\else
2653       \global\bbl@carg\chardef{l@#1}\@tempcnta
2654       \fi}}

```

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

```

2655 \def\bbl@input@texini#1{%
2656   \bbl@bsphack
2657   \bbl@exp{%

```

```

2658 \catcode`\\%=14 \catcode`\\\=0
2659 \catcode`\\{=1 \catcode`\\}=2
2660 \lowercase{\\InputIfFileExists{babel-#1.tex}{}}%
2661 \catcode`\\%=the\catcode`\relax
2662 \catcode`\\\=the\catcode`\\\relax
2663 \catcode`\\{=the\catcode`\relax
2664 \catcode`\\}=the\catcode`\relax}%
2665 \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.

```

2666 \def\bbl@iniline#1\bbl@iniline{%
2667 \ifnextchar[\bbl@inisect{\ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2668 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2669 \def\bbl@iniskip#1\@@{% if starts with ;
2670 \def\bbl@inistore#1=#2\@@{% full (default)
2671 \bbl@trim@def\bbl@tempa{#1}%
2672 \bbl@trim\toks@{#2}%
2673 \bbl@ifsamestring{\bbl@tempa}{\include}%
2674 {\bbl@read@subini{\the\toks@}}%
2675 {\bbl@xin@;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2676 \ifin@else
2677 \bbl@xin@{,identification/include.}%
2678 {,\bbl@section/\bbl@tempa}%
2679 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2680 \bbl@exp{%
2681 \\g@addto@macro\\bbl@inidata{%
2682 \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}%
2683 \fi}}
2684 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2685 \bbl@trim@def\bbl@tempa{#1}%
2686 \bbl@trim\toks@{#2}%
2687 \bbl@xin@{.identification.}{.\bbl@section.}%
2688 \ifin@
2689 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2690 \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}%
2691 \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.

```

2692 \def\bbl@loop@ini#1{%
2693 \loop
2694 \if T\ifeof#1 F\fi T\relax % Trick, because inside \loop
2695 \endlinechar\m@ne
2696 \read#1 to \bbl@line
2697 \endlinechar`^^M
2698 \ifx\bbl@line\empty\else
2699 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2700 \fi
2701 \repeat}

```

```

2702 %
2703 \def\bbl@read@subini#1{%
2704   \ifx\bbl@readsubstream\undefined
2705     \csname newread\endcsname\bbl@readsubstream
2706   \fi
2707   \openin\bbl@readsubstream=babel-#1.ini
2708   \ifeof\bbl@readsubstream
2709     \bbl@error{no-ini-file}{#1}{}{}%
2710   \else
2711     {\bbl@loop@ini\bbl@readsubstream}%
2712   \fi
2713   \closein\bbl@readsubstream}
2714 %
2715 \ifx\bbl@readstream\undefined
2716   \csname newread\endcsname\bbl@readstream
2717 \fi
2718 \def\bbl@read@ini#1#2{%
2719   \global\let\bbl@extend@ini@gobble
2720   \openin\bbl@readstream=babel-#1.ini
2721   \ifeof\bbl@readstream
2722     \bbl@error{no-ini-file}{#1}{}{}%
2723   \else
2724     % == Store ini data in \bbl@inidata ==
2725     \catcode\ =10 \catcode\ " =12
2726     \catcode\ [=12 \catcode\ ]=12 \catcode\ ==12 \catcode\ &=12
2727     \catcode\ ;=12 \catcode\ |=12 \catcode\ %=14 \catcode\ -=12
2728     \ifnum#2=\m@ne % Just for the info
2729       \edef\language\tag\bbl@metalang}%
2730     \fi
2731     \bbl@info{\ifnum#2=\m@ne Fetching locale name for tag\bbl@metalang
2732               \else Importing
2733                 \ifcase#2font and identification \or basic \fi
2734                 data for \language
2735               \fi\%
2736               from babel-#1.ini. Reported}%
2737     \ifnum#2<\@ne
2738       \global\let\bbl@inidata\@empty
2739       \let\bbl@inistore\bbl@inistore@min % Remember it's local
2740     \fi
2741     \def\bbl@section{identification}%
2742     \bbl@exp{%
2743       \\bbl@inistore tag.ini=#1\\@@
2744       \\bbl@inistore load.level=\ifnum#2<\@ne 0\else #2\fi\\@@}%
2745     \bbl@loop@ini\bbl@readstream
2746     % == Process stored data ==
2747     \ifnum#2=\m@ne
2748       \def\bbl@tempa##1 ##2\@{##1}% Get first name
2749       \def\bbl@elt###1##2##3{%
2750         \bbl@ifsamestring{identification/name.babel}{##1/##2}%
2751         {\edef\language{\bbl@tempa##3 \@}%
2752          \let\localename\language
2753          \bbl@id@assign
2754          \def\bbl@elt####1####2####3{}}%
2755         {}}%
2756       \bbl@inidata
2757     \fi
2758     \bbl@csarg\xdef{lini@\language}{#1}%
2759     \bbl@read@ini@aux
2760     % == 'Export' data ==
2761     \bbl@ini@exports{#2}%
2762     \global\bbl@csarg\let{inidata@\language}\bbl@inidata
2763     \global\let\bbl@inidata\@empty
2764     \bbl@exp{\\bbl@add@list\\bbl@ini@loaded{\language}}%

```



```

2765 \bbl@tglobal\bbl@ini@loaded
2766 \fi
2767 \closein\bbl@readstream}
2768 \def\bbl@read@ini@aux{%
2769 \let\bbl@savestrings\@empty
2770 \let\bbl@savetoday\@empty
2771 \let\bbl@savedate\@empty
2772 \def\bbl@elt##1##2##3{%
2773 \def\bbl@section{##1}%
2774 \in@{=date.}{=##1}% Find a better place
2775 \ifin@
2776 \bbl@ifunset{bbl@inikv@##1}%
2777 {\bbl@ini@calendar{##1}}%
2778 }%
2779 \fi
2780 \bbl@ifunset{bbl@inikv@##1}{}%
2781 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2782 \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.

```

2783 \def\bbl@extend@ini@aux#1{%
2784 \bbl@startcommands*{#1}{captions}%
2785 % Activate captions/... and modify exports
2786 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2787 \setlocalecaption{#1}{##1}{##2}}%
2788 \def\bbl@inikv@captions##1##2{%
2789 \bbl@ini@captions@aux{##1}{##2}}%
2790 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2791 \def\bbl@exportkey##1##2##3{%
2792 \bbl@ifunset{bbl@kv@##2}{%
2793 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2794 \bbl@exp{\global\let\<bbl@##1\language\>\<bbl@kv@##2>}}%
2795 \fi}}%
2796 % As with \bbl@read@ini, but with some changes
2797 \bbl@read@ini@aux
2798 \bbl@ini@exports\tw@
2799 % Update inidata@lang by pretending the ini is read.
2800 \def\bbl@elt##1##2##3{%
2801 \def\bbl@section{##1}%
2802 \bbl@iniline##2=##3\bbl@iniline}%
2803 \csname bbl@inidata@#1\endcsname
2804 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2805 \StartBabelCommands*{#1}{date}% And from the import stuff
2806 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2807 \bbl@savetoday
2808 \bbl@savedate
2809 \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections.

```

2810 \def\bbl@ini@calendar#1{%
2811 \lowercase{\def\bbl@tempa{=#1=}}%
2812 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2813 \bbl@replace\bbl@tempa{=date.}{}%
2814 \in@{.licr=}#1=%
2815 \ifin@
2816 \ifcase\bbl@engine
2817 \bbl@replace\bbl@tempa{.licr=}{}%
2818 \else
2819 \let\bbl@tempa\relax
2820 \fi
2821 \fi
2822 \ifx\bbl@tempa\relax\else
2823 \bbl@replace\bbl@tempa{=}{}%

```

```

2824 \ifx\babel@tempa\@empty\else
2825   \xdef\babel@calendars{\babel@calendars,\babel@tempa}%
2826 \fi
2827 \babel@exp{%
2828   \def\<babel@inikv@#1>####1####2{%
2829     \\babel@inidate####1...\relax{####2}{\babel@tempa}}}%
2830 \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 `\babel@inistore` above).

```

2831 \def\babel@renewinikv#1/#2\@#3{%
2832   \global\let\babel@extend@ini\babel@extend@ini@aux
2833   \edef\babel@tempa{\zap@space #1 \@empty}% section
2834   \edef\babel@tempb{\zap@space #2 \@empty}% key
2835   \babel@trim\toks@{#3}% value
2836   \babel@exp{%
2837     \edef\\babel@key@list{\babel@key@list \babel@tempa/\babel@tempb;}%
2838     \\g@addto@macro\\babel@inidata{%
2839       \\babel@elt{\babel@tempa}{\babel@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.

```

2840 \def\babel@exportkey#1#2#3{%
2841   \babel@ifunset{\babel@kv@#2}%
2842   {\babel@csarg\gdef{#1@\languagename}{#3}}%
2843   {\expandafter\ifx\csname babel@kv@#2\endcsname\@empty
2844     \babel@csarg\gdef{#1@\languagename}{#3}%
2845     \else
2846     \babel@exp{\global\let\<babel@#1@\languagename>\<babel@kv@#2>}%
2847     \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 `\babel@ini@exports` is called always (via `\babel@inisec`), while `\babel@after@ini` must be called explicitly after `\babel@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.

```

2848 \def\babel@iniwarning#1{%
2849   \babel@ifunset{\babel@kv@identification.warning#1}{}%
2850   {\babel@warning{%
2851     From babel-\babel@cs{lini@\languagename}.ini:\\%
2852     \babel@cs{@kv@identification.warning#1}\\%
2853     Reported}}}
2854 %
2855 \let\babel@release@transforms\@empty
2856 \let\babel@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).

```

2857 \def\babel@ini@exports#1{%
2858   % Identification always exported
2859   \babel@iniwarning{%
2860   \ifcase\babel@engine
2861     \babel@iniwarning{.pdfLaTeX}%

```

```

2862 \or
2863   \bbl@iniwarning{.lualatex}%
2864 \or
2865   \bbl@iniwarning{.xelatex}%
2866 \fi%
2867 \bbl@exportkey{llevel}{identification.load.level}{}%
2868 \bbl@exportkey{elname}{identification.name.english}{}%
2869 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2870   {\csname bbl@elname@\languagename\endcsname}}%
2871 \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2872 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2873 \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2874 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2875 \bbl@exportkey{esname}{identification.script.name}{}%
2876 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2877   {\csname bbl@esname@\languagename\endcsname}}%
2878 \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
2879 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2880 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2881 \bbl@exportkey{vbcpl}{identification.variant.tag.bcp47}{}%
2882 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2883 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2884 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2885 % Also maps bcp47 -> languagename
2886 \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{\languagename}%
2887 \ifcase\bbl@engine\or
2888   \directlua{%
2889     Babel.locale_props[\the\bbl@cs{id@@\languagename}].script
2890     = '\bbl@cl{sbcpl}'}%
2891 \fi
2892 % Conditional
2893 \ifnum#1>\z@      % -1 or 0 = only info, 1 = basic, 2 = (re)new
2894   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2895   \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
2896   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2897   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2898   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
2899   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2900   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2901   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2902   \bbl@exportkey{intsp}{typography.intraspace}{}%
2903   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2904   \bbl@exportkey{chrng}{characters.ranges}{}%
2905   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2906   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2907   \ifnum#1=\tw@      % only (re)new
2908     \bbl@exportkey{rqtex}{identification.require.babel}{}%
2909     \bbl@tglobal\bbl@savetoday
2910     \bbl@tglobal\bbl@savestate
2911     \bbl@savestrings
2912   \fi
2913 \fi}

```

4.20. Processing keys in ini

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

```

2914 \def\bbl@inikv#1#2{%      key=value
2915   \toks@{#2}%             This hides #'s from ini values
2916   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2917 \let\bbl@inikv@identification\bbl@inikv
2918 \let\bbl@inikv@date\bbl@inikv

```

```
2919 \let\bbl@inikv@typography\bbl@inikv
2920 \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`.

```
2921 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\@empty x-\fi}
2922 \def\bbl@inikv@characters#1#2{%
2923   \bbl@ifsamestring{#1}{casing}% e.g., casing = uV
2924   {\bbl@exp{%
2925     \\g@addto@macro\\bbl@release@casing{%
2926       \\bbl@casemapping}{\language}{\unexpanded{#2}}}%
2927   {\in@{casing.}{#1}% e.g., casing.Uv = uV
2928   \ifin@
2929     \lowercase{\def\bbl@tempb{#1}}%
2930     \bbl@replace\bbl@tempb{casing.}{}%
2931     \bbl@exp{\\g@addto@macro\\bbl@release@casing{%
2932       \\bbl@casemapping
2933       {\\bbl@maybextx\bbl@tempb}{\language}{\unexpanded{#2}}}%
2934   \else
2935     \bbl@inikv{#1}{#2}%
2936   \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’.

```
2937 \def\bbl@inikv@counters#1#2{%
2938   \bbl@ifsamestring{#1}{digits}%
2939   {\bbl@error{digits-is-reserved}{}}{}%
2940   }%
2941   \def\bbl@tempc{#1}%
2942   \bbl@trim@def{\bbl@tempb*}{#2}%
2943   \in@{.1$}{#1$}%
2944   \ifin@
2945     \bbl@replace\bbl@tempc{.1}{}%
2946     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}{%
2947       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2948   \fi
2949   \in@{.F.}{#1}%
2950   \ifin@\else\in@{.S.}{#1}\fi
2951   \ifin@
2952     \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
2953   \else
2954     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2955     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
2956     \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
2957   \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.

```
2958 \ifcase\bbl@engine
2959   \bbl@csarg\def{inikv@captions.licr}#1#2{%
2960     \bbl@ini@captions@aux{#1}{#2}}
2961 \else
2962   \def\bbl@inikv@captions#1#2{%
2963     \bbl@ini@captions@aux{#1}{#2}}
2964 \fi
```

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

```
2965 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2966   \bbl@replace\bbl@tempa{.template}{}%
2967   \def\bbl@toreplace{#1}{}}%
2968   \bbl@replace\bbl@toreplace[ ]{\nobreakspace}}%
```

```

2969 \bbl@replace\bbl@toreplace{[[]]{\csname}%
2970 \bbl@replace\bbl@toreplace{[[]]{\csname the}%
2971 \bbl@replace\bbl@toreplace{[[]]{name\endcsname}}}%
2972 \bbl@replace\bbl@toreplace{[[]]{\endcsname}}}%
2973 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
2974 \ifin@
2975 \@nameuse{bbl@patch\bbl@tempa}%
2976 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2977 \fi
2978 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
2979 \ifin@
2980 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
2981 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
2982 \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\\language}%
2983 {[\fnum@\bbl@tempa]}%
2984 {\\\@nameuse{bbl@\bbl@tempa fmt@\\\language}}}}%
2985 \fi}
2986 %
2987 \def\bbl@ini@captions@aux#1#2{%
2988 \bbl@trim@def\bbl@tempa{#1}%
2989 \bbl@xin@{.template}{\bbl@tempa}%
2990 \ifin@
2991 \bbl@ini@captions@template{#2}\language
2992 \else
2993 \bbl@ifblank{#2}%
2994 {\bbl@exp{%
2995 \toks@{\\\bbl@nocaption{\bbl@tempa name}{\language\bbl@tempa name}}}%
2996 {\bbl@trim\toks@{#2}}%
2997 \bbl@exp{%
2998 \\\bbl@add\\\bbl@savestrings{%
2999 \\\SetString\<\bbl@tempa name>{\the\toks@}}%
3000 \toks@\expandafter{\bbl@captionslist}%
3001 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3002 \ifin@\else
3003 \bbl@exp{%
3004 \\\bbl@add\<bbl@extracaps@\language>{\<\bbl@tempa name>}%
3005 \\\bbl@tglobal\<bbl@extracaps@\language>}%
3006 \fi
3007 \fi}

```

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

```

3008 \def\bbl@list@the{%
3009 part,chapter,section,subsection,subsubsection,paragraph,%
3010 subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3011 table,page,footnote,mpfootnote,mpfn}
3012 %
3013 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3014 \bbl@ifunset{bbl@map@#1@\language}%
3015 {\@nameuse{#1}}%
3016 {\@nameuse{bbl@map@#1@\language}}}
3017 %
3018 \def\bbl@map@lbl#1{% #1:a sign, eg, .
3019 \ifincsname#1\else
3020 \bbl@ifunset{bbl@map@@#1@\language}%
3021 {#1}%
3022 {\@nameuse{bbl@map@@#1@\language}}%
3023 \fi}
3024 %
3025 \def\bbl@inikv@labels#1#2{%
3026 \in@{.map}{#1}%
3027 \ifin@
3028 \in@{,dot.map,}{,#1,}%
3029 \ifin@

```

```

3030     \global\@namedef{bbl@map@.@@\language name}{#2}%
3031     \fi
3032     \ifx\bbl@KVP@labels\@nnil\else
3033       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3034       \ifin@
3035         \def\bbl@tempc{#1}%
3036         \bbl@replace\bbl@tempc{.map}{}%
3037         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3038         \bbl@exp{%
3039           \gdef\<bbl@map@\bbl@tempc @\language name>%
3040             {\ifin@<#2>\else\\localedcounter{#2}\fi}}%
3041         \bbl@foreach\bbl@list@the{%
3042           \bbl@ifunset{the##1}{}%
3043             {\bbl@ncarg\let\bbl@tempd{the##1}%
3044               \bbl@exp{%
3045                 \\bbl@sreplace\<the##1>%
3046                   {\<\bbl@tempc>{##1}}%
3047                   {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3048                   \\bbl@sreplace\<the##1>%
3049                   {\<\@empty @\bbl@tempc>\<c@##1>}%
3050                   {\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3051                   \\bbl@sreplace\<the##1>%
3052                   {\\\csname @\bbl@tempc\\endcsname\<c@##1>}%
3053                   {\{\bbl@map@cnt{\bbl@tempc}{##1}}}}%
3054           \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3055             \bbl@exp{\gdef\<the##1>{\{the##1\}}}%
3056           \fi}}%
3057     \fi
3058     \fi
3059 %
3060 \else
3061 % The following code is still under study. You can test it and make
3062 % suggestions. E.g., enumerate.2 = ([enumi]).([enumii]). It's
3063 % language dependent.
3064 \in@{enumerate.}{#1}%
3065 \ifin@
3066   \def\bbl@tempa{#1}%
3067   \bbl@replace\bbl@tempa{enumerate.}{}%
3068   \def\bbl@toreplace{#2}%
3069   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3070   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3071   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3072   \toks@\expandafter{\bbl@toreplace}%
3073   \bbl@exp{%
3074     \\bbl@add\<extras\language name>{%
3075       \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3076       \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3077     \\bbl@tglobal\<extras\language name>}%
3078   \fi
3079 \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.

```

3080 \def\bbl@chapttype{chapter}
3081 \ifx\@makechapterhead\@undefined
3082   \let\bbl@patchchapter\relax
3083 \else\ifx\thechapter\@undefined
3084   \let\bbl@patchchapter\relax
3085 \else\ifx\ps@headings\@undefined
3086   \let\bbl@patchchapter\relax
3087 \else

```

```

3088 \def\bbbl@patchchapter{%
3089   \global\let\bbbl@patchchapter\relax
3090   \gdef\bbbl@chfmt{%
3091     \bbbl@ifunset{bbbl@bbbl@chapttype fmt@\languagename}%
3092     {\@chapapp\space\thechapter}%
3093     {\@nameuse{bbbl@bbbl@chapttype fmt@\languagename}}}%
3094   \bbbl@add\appendix{\def\bbbl@chapttype{appendix}}% Not harmful, I hope
3095   \bbbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbbl@chfmt}%
3096   \bbbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbbl@chfmt}%
3097   \bbbl@sreplace\makechapterhead{\@chapapp\space\thechapter}{\bbbl@chfmt}%
3098   \bbbl@tglobal\appendix
3099   \bbbl@tglobal\ps@headings
3100   \bbbl@tglobal\chaptermark
3101   \bbbl@tglobal\@makechapterhead}
3102   \let\bbbl@patchappendix\bbbl@patchchapter
3103 \fi\fi\fi
3104 \ifx\@part\undefined
3105   \let\bbbl@patchpart\relax
3106 \else
3107   \def\bbbl@patchpart{%
3108     \global\let\bbbl@patchpart\relax
3109     \gdef\bbbl@partformat{%
3110       \bbbl@ifunset{bbbl@partfmt@\languagename}%
3111       {\partname\nobreakspace\thepart}%
3112       {\@nameuse{bbbl@partfmt@\languagename}}}%
3113     \bbbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbbl@partformat}%
3114     \bbbl@tglobal\@part}
3115 \fi

```

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

```

3116 \let\bbbl@calendar\@empty
3117 \DeclareRobustCommand\localedate[1][\bbbl@localedate{#1}]
3118 \def\bbbl@localedate#1#2#3#4{%
3119   \begingroup
3120     \edef\bbbl@they{#2}%
3121     \edef\bbbl@them{#3}%
3122     \edef\bbbl@thed{#4}%
3123     \edef\bbbl@tempe{%
3124       \bbbl@ifunset{bbbl@calpr@\languagename}{\bbbl@cl{calpr}},%
3125       #1}%
3126     \bbbl@exp{\lowercase{\edef\bbbl@tempe{\bbbl@tempe}}}%
3127     \bbbl@replace\bbbl@tempe{ }{}%
3128     \bbbl@replace\bbbl@tempe{convert}{convert=}%
3129     \let\bbbl@ld@calendar\@empty
3130     \let\bbbl@ld@variant\@empty
3131     \let\bbbl@ld@convert\relax
3132     \def\bbbl@tempb##1=##2\@{\@namedef{bbbl@ld##1}{##2}}%
3133     \bbbl@foreach\bbbl@tempe{\bbbl@tempb##1\@}%
3134     \bbbl@replace\bbbl@ld@calendar{gregorian}{}%
3135     \ifx\bbbl@ld@calendar\@empty\else
3136       \ifx\bbbl@ld@convert\relax\else
3137         \babelcalendar[\bbbl@they-\bbbl@them-\bbbl@thed]%
3138         {\bbbl@ld@calendar}\bbbl@they\bbbl@them\bbbl@thed
3139       \fi
3140     \fi
3141     \@nameuse{bbbl@precalendar}% Remove, e.g., +, -civil (-ca-islamic)
3142     \edef\bbbl@calendar{% Used in \month..., too
3143       \bbbl@ld@calendar
3144       \ifx\bbbl@ld@variant\@empty\else
3145         .\bbbl@ld@variant
3146       \fi}%
3147     \bbbl@cased

```

```

3148     {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3149     \bbl@they\bbl@them\bbl@thed}%
3150 \endgroup}
3151 %
3152 \def\bbl@printdate#1{%
3153   \@ifnextchar[{\bbl@printdate@i{#1}}{\bbl@printdate@i{#1}[]}]
3154 \def\bbl@printdate@i#1[#2]#3#4#5{%
3155   \bbl@usedategrouptrue
3156   \@nameuse{bbl@ensure@#1}{\localedate[#2]{#3}{#4}{#5}}
3157 %
3158 % e.g.: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3159 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{%
3160   \bbl@trim@def\bbl@tempa{#1.#2}%
3161   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3162   {\bbl@trim@def\bbl@tempa{#3}%
3163     \bbl@trim\toks@{#5}%
3164     \@temptokena\expandafter{\bbl@savodate}%
3165     \bbl@exp{% Reverse order - in ini last wins
3166       \def\\bbl@savodate{%
3167         \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3168         \the\@temptokena}}%
3169   {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3170     {\lowercase{\def\bbl@tempb{#6}}%
3171     \bbl@trim@def\bbl@toreplace{#5}%
3172     \bbl@TG@@date
3173     \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3174     \if\bbl@savetoday\@empty
3175     \bbl@exp{%
3176       \\AfterBabelCommands{%
3177         \gdef\<\languagename date>{\protect\<\languagename date >}%
3178         \gdef\<\languagename date >{\bbl@printdate{\languagename}}}%
3179     \def\\bbl@savetoday{%
3180       \\SetString\\today{%
3181         \<\languagename date>[convert]%
3182         {\the\year}{\the\month}{\the\day}}}%
3183     \fi}%
3184   {}}}}

```

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).

```

3185 \let\bbl@calendar\@empty
3186 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3187   \@nameuse{bbl@ca@#2}#1\@@}
3188 \newcommand\babelDateSpace{\nobreakspace}
3189 \newcommand\babelDateDot{.\@}
3190 \newcommand\babelDated[1][{\number#1}]
3191 \newcommand\babelDatedd[1][{\ifnum#1<10 0\fi\number#1}]
3192 \newcommand\babelDateM[1][{\number#1}]
3193 \newcommand\babelDateMM[1][{\ifnum#1<10 0\fi\number#1}]
3194 \newcommand\babelDateMMMM[1][{%
3195   \csname month\romannumeral#1\bbl@calendar name\endcsname}]%
3196 \newcommand\babelDatey[1][{\number#1}]%
3197 \newcommand\babelDateyy[1][{%
3198   \ifnum#1<10 0\number#1 %
3199   \else\ifnum#1<100 \number#1 %
3200   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3201   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3202   \else
3203     \bbl@error{limit-two-digits}{\number#1}{\number#1}%
3204   \fi\fi\fi\fi}

```



```

3205 \newcommand\BabelDateyyyy[1]{\number#1}
3206 \newcommand\BabelDateU[1]{\number#1}%
3207 \def\bb@l@replace@finish@iii#1{%
3208   \bb@l@exp{\def\#1###1###2###3{\the\toks@}}
3209 \def\bb@l@TG@date{%
3210   \bb@l@replace\bb@l@toreplace{[ ]}{\BabelDateSpace{}}%
3211   \bb@l@replace\bb@l@toreplace{[. ]}{\BabelDateDot{}}%
3212   \bb@l@replace\bb@l@toreplace{[y]}{\BabelDatey{###1}}%
3213   \bb@l@replace\bb@l@toreplace{[y]}{\bb@l@datecncr{###1|}}%
3214   \bb@l@replace\bb@l@toreplace{[yy]}{\BabelDateyy{###1}}%
3215   \bb@l@replace\bb@l@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3216   \bb@l@replace\bb@l@toreplace{[M]}{\BabelDateM{###2}}%
3217   \bb@l@replace\bb@l@toreplace{[M]}{\bb@l@datecncr{###2|}}%
3218   \bb@l@replace\bb@l@toreplace{[MM]}{\BabelDateMM{###2}}%
3219   \bb@l@replace\bb@l@toreplace{[MMMM]}{\BabelDateMMMM{###2}}%
3220   \bb@l@replace\bb@l@toreplace{[d]}{\BabelDated{###3}}%
3221   \bb@l@replace\bb@l@toreplace{[d]}{\bb@l@datecncr{###3|}}%
3222   \bb@l@replace\bb@l@toreplace{[dd]}{\BabelDatedd{###3}}%
3223   \bb@l@replace\bb@l@toreplace{[U]}{\BabelDateU{###1}}%
3224   \bb@l@replace\bb@l@toreplace{[U]}{\bb@l@datecncr{###1|}}%
3225   \bb@l@replace@finish@iii\bb@l@toreplace}
3226 \def\bb@l@datecncr{\expandafter\bb@l@xdatecncr\expandafter}
3227 \def\bb@l@xdatecncr[#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.

```

3228 \AddToHook{begindocument/before}{%
3229   \let\bb@l@normalsf\normalsfcodes
3230   \let\normalsfcodes\relax}
3231 \AtBeginDocument{%
3232   \ifx\bb@l@normalsf\@empty
3233     \ifnum\sffcode`\.=\@m
3234       \let\normalsfcodes\frenchspacing
3235     \else
3236       \let\normalsfcodes\nonfrenchspacing
3237     \fi
3238   \else
3239     \let\normalsfcodes\bb@l@normalsf
3240   \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 `\bb@l@transforms@aux` ...`\relax`, and stores them in `\bb@l@release@transforms`. However, since building a list enclosed in braces isn't trivial, the replacements are added after a comma, and then `\bb@l@transforms@aux` adds the braces.

```

3241 \bb@l@csarg\let{inikv@transforms.prehyphenation}\bb@l@inikv
3242 \bb@l@csarg\let{inikv@transforms.posthyphenation}\bb@l@inikv
3243 \def\bb@l@transforms@aux#1#2#3#4,#5\relax{%
3244   #1|#2|#3|#4|#5}
3245 \begingroup
3246   \catcode`\%=12
3247   \catcode`\&=14
3248   \gdef\bb@l@transforms#1#2#3{&%
3249     \directlua{
3250       local str = [==[#2]==]
3251       str = str:gsub('%.%d+%.%d+$', '')
3252       token.set_macro('babeltempa', str)
3253     }&%
3254   \def\babeltempc{&%
3255   \bb@l@xin@{,\babeltempa,},{,\bb@l@KVP@transforms,}&%

```

```

3256 \ifin\else
3257   \bbl@xin@{: \babeltempa,}{, \bbl@KVP@transforms,}&%
3258 \fi
3259 \ifin@
3260   \bbl@foreach\bbl@KVP@transforms{&%
3261     \bbl@xin@{: \babeltempa,}{, ##1,}&%
3262     \ifin@ &% font:font:transform syntax
3263     \directlua{
3264       local t = {}
3265       for m in string.gmatch('##1'..' ':'', '(.):') do
3266         table.insert(t, m)
3267       end
3268       table.remove(t)
3269       token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3270     }&%
3271   \fi}&%
3272 \in@{.0$}{#2$}&%
3273 \ifin@
3274   \directlua{&% (\attribute) syntax
3275     local str = string.match([[ \bbl@KVP@transforms]],
3276       '^(([^%(-)]^)[^)]-\babeltempa')
3277     if str == nil then
3278       token.set_macro('babeltempb', '')
3279     else
3280       token.set_macro('babeltempb', ', attribute=' .. str)
3281     end
3282   }&%
3283 \toks@{#3}&%
3284 \bbl@exp{&%
3285   \\g@addto@macro\\bbl@release@transforms{&%
3286     \relax &% Closes previous \bbl@transforms@aux
3287     \\bbl@transforms@aux
3288     \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3289     {\languagename}{\the\toks@}}&%
3290 \else
3291   \g@addto@macro\bbl@release@transforms{, {#3}}&%
3292 \fi
3293 \fi}
3294 \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.

```

3295 \def\bbl@provide@lsys#1{%
3296   \bbl@ifunset{bbl@lname@#1}%
3297     {\bbl@load@info{#1}}%
3298   }%
3299 \bbl@csarg\let{lsys@#1}\@empty
3300 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{%
3301   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{%
3302     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3303     \bbl@ifunset{bbl@lname@#1}{%
3304       {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3305     \ifcase\bbl@engine\or\or
3306       \bbl@ifunset{bbl@prehc@#1}{%
3307         {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}}%
3308         }%
3309         {\ifx\bbl@xenohyph\@undefined
3310           \global\let\bbl@xenohyph\bbl@xenohyph@d
3311           \ifx\AtBeginDocument\@notprerr

```



```

3361 \bbl@ifunset{bbl@cntr@#1@\languagename}%
3362 {#2}%
3363 {\bbl@cs{cntr@#1@\languagename}{#2}}
3364 \def\bbl@localecntr#1#2{\localenumerat{#2}{#1}}
3365 \newcommand\localecounter[2]{%
3366 \expandafter\bbl@localecntr
3367 \expandafter{\number\csname c@#2\endcsname}{#1}}
3368 \def\bbl@alphanumeric#1#2{%
3369 \expandafter\bbl@alphanumeric@i\number#2 76543210\@@{#1}}
3370 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8\@@#9{%
3371 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3372 \bbl@alphanumeric@ii{#9}00000#1\or
3373 \bbl@alphanumeric@ii{#9}00000#1#2\or
3374 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3375 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3376 \bbl@alphnum@invalid{>9999}%
3377 \fi}
3378 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3379 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3380 {\bbl@cs{cntr@#1.4@\languagename}#5%
3381 \bbl@cs{cntr@#1.3@\languagename}#6%
3382 \bbl@cs{cntr@#1.2@\languagename}#7%
3383 \bbl@cs{cntr@#1.1@\languagename}#8%
3384 \ifnum#6#7#8>\z@
3385 \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3386 {\bbl@cs{cntr@#1.S.321@\languagename}}%
3387 \fi}%
3388 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}
3389 \def\bbl@alphnum@invalid#1{%
3390 \bbl@error{alphabetic-too-large}{#1}{}}

```

4.24. Casing

```

3391 \newcommand\BabelUppercaseMapping[3]{%
3392 \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3393 \newcommand\BabelTitlecaseMapping[3]{%
3394 \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3395 \newcommand\BabelLowercaseMapping[3]{%
3396 \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

The parser for casing and casing.<variant>.
3397 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3398 \def\bbl@utftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3399 \else
3400 \def\bbl@utftocode#1{\expandafter`\string#1}
3401 \fi
3402 \def\bbl@casemapping#1#2#3{% 1:variant
3403 \def\bbl@tempa##1 ##2{% Loop
3404 \bbl@casemapping@i{##1}%
3405 \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3406 \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3407 \def\bbl@tempe{0}% Mode (upper/lower...)
3408 \def\bbl@tempc{#3 }% Casing list
3409 \expandafter\bbl@tempa\bbl@tempc\@empty}
3410 \def\bbl@casemapping@i#1{%
3411 \def\bbl@tempb{#1}%
3412 \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3413 \@nameuse{regex_replace_all:nnN}%
3414 {[[\x{c0}-\x{ff}][\x{80}-\x{bf}]*]{\0}}\bbl@tempb
3415 \else
3416 \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb
3417 \fi
3418 \expandafter\bbl@casemapping@ii\bbl@tempb\@@}
3419 \def\bbl@casemapping@ii#1#2#3\@@{%

```

```

3420 \in{#1#3}{<>}% i.e., if <u>, <l>, <t>
3421 \ifin@
3422 \edef\bbl@tempe{%
3423 \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3424 \else
3425 \ifcase\bbl@tempe\relax
3426 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3427 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3428 \or
3429 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3430 \or
3431 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3432 \or
3433 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3434 \fi
3435 \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.

```

3436 \def\bbl@localeinfo#1#2{%
3437 \bbl@ifunset{bbl@info@#2}{#1}%
3438 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @\languagename}{#1}%
3439 {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}%
3440 \newcommand\localeinfo[1]{%
3441 \ifx*#1\@empty
3442 \bbl@afterelse\bbl@localeinfo}%
3443 \else
3444 \bbl@localeinfo
3445 {\bbl@error{no-ini-info}{}}{}%
3446 {#1}%
3447 \fi}
3448 % \@namedef{bbl@info@name.locale}{lcname}
3449 \@namedef{bbl@info@tag.ini}{lini}
3450 \@namedef{bbl@info@name.english}{elname}
3451 \@namedef{bbl@info@name.opentype}{lname}
3452 \@namedef{bbl@info@tag.bcp47}{tbc}
3453 \@namedef{bbl@info@language.tag.bcp47}{lbc}
3454 \@namedef{bbl@info@tag.opentype}{lotf}
3455 \@namedef{bbl@info@script.name}{esname}
3456 \@namedef{bbl@info@script.name.opentype}{sname}
3457 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3458 \@namedef{bbl@info@script.tag.opentype}{sotf}
3459 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3460 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3461 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3462 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3463 \@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.

```

3464 <<{*More package options}>> ≡
3465 \DeclareOption{ensureinfo=off}{}
3466 <</More package options>>
3467 \let\BabelEnsureInfo\relax

```

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

```

3468 \newcommand\getlocaleproperty{%
3469 \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3470 \def\bbl@getproperty@s#1#2#3{%
3471 \let#1\relax
3472 \def\bbl@elt##1##2##3{%
3473 \bbl@ifsamestring{##1/##2}{#3}%

```

```

3474     {\providecommand#1{##3}%
3475     \def\bbl@elt###1###2###3{}}%
3476     {}}%
3477     \bbl@cs{inidata@#2}}%
3478 \def\bbl@getproperty@x#1#2#3{%
3479     \bbl@getproperty@#1}{#2}{#3}%
3480     \ifx#1\relax
3481     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3482     \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`.

```

3483 \let\bbl@ini@loaded\@empty
3484 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3485 \def\ShowLocaleProperties#1{%
3486     \typeout{}}%
3487     \typeout{*** Properties for language '#1' ***}%
3488     \def\bbl@elt###1###2###3{\typeout{##1/##2 = \unexpanded{##3}}}%
3489     \@nameuse{\bbl@inidata@#1}%
3490     \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.

```

3491 \newif\ifbbl@bcpallowed
3492 \bbl@bcpallowedfalse
3493 \def\bbl@autoload@options{@import}
3494 \def\bbl@provide@locale{%
3495     \ifx\babelprovide\undefined
3496     \bbl@error{base-on-the-fly}{}{}%
3497     \fi
3498     \let\bbl@auxname\languagename
3499     \ifbbl@bcptoname
3500     \bbl@ifunset{\bbl@bcp@map@\languagename}{}% Move uplevel??
3501     {\edef\languagename{\@nameuse{\bbl@bcp@map@\languagename}}}%
3502     \let\localename\languagename}%
3503     \fi
3504     \ifbbl@bcpallowed
3505     \expandafter\ifx\csname date\languagename\endcsname\relax
3506     \expandafter
3507     \bbl@bcplookup\languagename-\@empty-\@empty-\@empty@@
3508     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
3509     \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
3510     \let\localename\languagename
3511     \expandafter\ifx\csname date\languagename\endcsname\relax
3512     \let\bbl@initoload\bbl@bcp
3513     \bbl@exp{\babelprovide[\bbl@autoload@bcptoptions]{\languagename}}%
3514     \let\bbl@initoload\relax
3515     \fi
3516     \bbl@csarg\xdef{\bbl@bcp@\localename}%
3517     \fi
3518     \fi
3519     \fi
3520     \expandafter\ifx\csname date\languagename\endcsname\relax
3521     \IfFileExists{babel-\languagename.tex}%
3522     {\bbl@exp{\babelprovide[\bbl@autoload@options]{\languagename}}}%
3523     {}}%

```

```
3524 \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`.

```
3525 \providecommand\BCPdata{}
3526 \ifx\renewcommand\undefined\else
3527 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty\@empty\@empty}
3528 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3529 \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3530 {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3531 {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3532 \def\bbl@bcpdata@ii#1#2{%
3533 \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3534 {\bbl@error{unknown-ini-field}{#1}{}}}%
3535 {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3536 {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}}
3537 \fi
3538 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3539 \@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.

```
3540 \newcommand\babeladjust[1]{%
3541 \bbl@forkv{#1}{%
3542 \bbl@ifunset{bbl@ADJ@##1@##2}%
3543 {\bbl@cs{ADJ@##1}{##2}}%
3544 {\bbl@cs{ADJ@##1@##2}}}
3545 %
3546 \def\bbl@adjust@lua#1#2{%
3547 \ifvmode
3548 \ifnum\currentgrouplevel=\z@
3549 \directlua{ Babel.#2 }%
3550 \expandafter\expandafter\expandafter\@gobble
3551 \fi
3552 \fi
3553 {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3554 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3555 \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3556 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3557 \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3558 \@namedef{bbl@ADJ@bidi.text@on}{%
3559 \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3560 \@namedef{bbl@ADJ@bidi.text@off}{%
3561 \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3562 \@namedef{bbl@ADJ@bidi.math@on}{%
3563 \let\bbl@noamsmath\@empty}
3564 \@namedef{bbl@ADJ@bidi.math@off}{%
3565 \let\bbl@noamsmath\relax}
3566 %
3567 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3568 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3569 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3570 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3571 %
3572 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3573 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3574 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3575 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
```

```

3576 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3577   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3578 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3579   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3580 \@namedef{bbl@ADJ@justify.arabic@on}{%
3581   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3582 \@namedef{bbl@ADJ@justify.arabic@off}{%
3583   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3584 %
3585 \def\bbl@adjust@layout#1{%
3586   \ifvmode
3587     #1%
3588     \expandafter\@gobble
3589   \fi
3590   {\bbl@error{layout-only-vertical}{}{}}}% Gobbled if everything went ok.
3591 \@namedef{bbl@ADJ@layout.tabular@on}{%
3592   \ifnum\bbl@tabular@mode=\tw@
3593     \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3594   \else
3595     \chardef\bbl@tabular@mode\@ne
3596   \fi}
3597 \@namedef{bbl@ADJ@layout.tabular@off}{%
3598   \ifnum\bbl@tabular@mode=\tw@
3599     \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3600   \else
3601     \chardef\bbl@tabular@mode\z@
3602   \fi}
3603 \@namedef{bbl@ADJ@layout.lists@on}{%
3604   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3605 \@namedef{bbl@ADJ@layout.lists@off}{%
3606   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3607 %
3608 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3609   \bbl@bcppallowedtrue}
3610 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3611   \bbl@bcppallowedfalse}
3612 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3613   \def\bbl@bcpp@prefix{#1}}
3614 \def\bbl@bcpp@prefix{bcp47-}
3615 \@namedef{bbl@ADJ@autoload.options}#1{%
3616   \def\bbl@autoload@options{#1}}
3617 \def\bbl@autoload@bcppoptions{import}
3618 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3619   \def\bbl@autoload@bcppoptions{#1}}
3620 \newif\ifbbl@bcptoname
3621 %
3622 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3623   \bbl@bcptonamettrue}
3624 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3625   \bbl@bcptonamefalse}
3626 %
3627 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3628   \directlua{ Babel.ignore_pre_char = function(node)
3629     return (node.lang == \the\csname \l@nohyphenation\endcsname)
3630   end }}
3631 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3632   \directlua{ Babel.ignore_pre_char = function(node)
3633     return false
3634   end }}
3635 %
3636 \@namedef{bbl@ADJ@interchar.disable@nohyphenation}{%
3637   \def\bbl@ignoreinterchar{%
3638     \ifnum\language=\l@nohyphenation

```



```

3639     \expandafter@gobble
3640     \else
3641     \expandafter@firstofone
3642     \fi}}
3643 \namedef{bbl@ADJ@interchar.disable@off}{%
3644   \let\bbl@ignoreinterchar@firstofone}
3645 %
3646 \namedef{bbl@ADJ@select.write@shift}{%
3647   \let\bbl@restorelastskip\relax
3648   \def\bbl@savelastskip{%
3649     \let\bbl@restorelastskip\relax
3650     \ifvmode
3651       \ifdim\lastskip=\z@
3652         \let\bbl@restorelastskip\nobreak
3653       \else
3654         \bbl@exp{%
3655           \def\\bbl@restorelastskip{%
3656             \skip@=\the\lastskip
3657             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3658         \fi
3659       \fi}}
3660 \namedef{bbl@ADJ@select.write@keep}{%
3661   \let\bbl@restorelastskip\relax
3662   \let\bbl@savelastskip\relax}
3663 \namedef{bbl@ADJ@select.write@omit}{%
3664   \AddBabelHook{babel-select}{beforestart}{%
3665     \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3666   \let\bbl@restorelastskip\relax
3667   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3668 \namedef{bbl@ADJ@select.encoding@off}{%
3669   \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.

```

3670 <<{*More package options}>> ≡
3671 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3672 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3673 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3674 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3675 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3676 <</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.

```

3677 \bbl@trace{Cross referencing macros}
3678 \ifx\bbl@opt@safe\@empty\else % i.e., if 'ref' and/or 'bib'
3679   \def\@newl@bel#1#2#3{%
3680     {\@safe@activestrue
3681       \bbl@ifunset{#1@#2}%
3682       \relax
3683       {\gdef\@multiplelabels{%

```

```

3684     \latex@warning@no@line{There were multiply-defined labels}}%
3685     \latex@warning@no@line{Label `#2' multiply defined}}%
3686     \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.

```

3687 \CheckCommand*\@testdef[3]{%
3688   \def\reserved@a{#3}%
3689   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3690   \else
3691     \@tempswatru
3692   \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 `\@newlabel` 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.

```

3693 \def\@testdef#1#2#3{%
3694   \@safe@activestru
3695   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3696   \def\bbl@tempb{#3}%
3697   \@safe@activestru
3698   \ifx\bbl@tempa\relax
3699   \else
3700     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3701   \fi
3702   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3703   \ifx\bbl@tempa\bbl@tempb
3704   \else
3705     \@tempswatru
3706   \fi}
3707 \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.

```

3708 \bbl@xin@{R}\bbl@opt@safe
3709 \ifin@
3710 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3711 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3712 {\expandafter\strip@prefix\meaning\ref}%
3713 \ifin@
3714 \bbl@redefine\@kernel@ref#1{%
3715   \@safe@activestru\org@@kernel@ref{#1}\@safe@activestru}
3716 \bbl@redefine\@kernel@pageref#1{%
3717   \@safe@activestru\org@@kernel@pageref{#1}\@safe@activestru}
3718 \bbl@redefine\@kernel@sref#1{%
3719   \@safe@activestru\org@@kernel@sref{#1}\@safe@activestru}
3720 \bbl@redefine\@kernel@spageref#1{%
3721   \@safe@activestru\org@@kernel@spageref{#1}\@safe@activestru}
3722 \else
3723 \bbl@redefineroobust\ref#1{%
3724   \@safe@activestru\org@ref{#1}\@safe@activestru}
3725 \bbl@redefineroobust\pageref#1{%
3726   \@safe@activestru\org@pageref{#1}\@safe@activestru}
3727 \fi
3728 \else
3729 \let\org@ref\ref
3730 \let\org@pageref\pageref
3731 \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.

```
3732 \bbl@xin@{B}\bbl@opt@safe
3733 \ifin@
3734 \bbl@redefine\@citex[#1]#2{%
3735   \@safe@activestruedef\bbl@tempa{#2}\@safe@activesfalse
3736   \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.)

```
3737 \AtBeginDocument{%
3738   \ifpackageloaded{natbib}{%
3739     \def\@citex[#1][#2]#3{%
3740       \@safe@activestruedef\bbl@tempa{#3}\@safe@activesfalse
3741       \org@citex[#1][#2]{\bbl@tempa}}%
3742   }{}}
```

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

```
3743 \AtBeginDocument{%
3744   \ifpackageloaded{cite}{%
3745     \def\@citex[#1]#2{%
3746       \@safe@activestruedef\org@citex[#1][#2]\@safe@activesfalse}%
3747   }{}}
```

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

```
3748 \bbl@redefine\nocite#1{%
3749   \@safe@activestruedef\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@activestruedef` 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.

```
3750 \bbl@redefine\bibcite{%
3751   \bbl@cite@choice
3752   \bibcite}
```

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

```
3753 \def\bbl@bibcite#1#2{%
3754   \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.

```
3755 \def\bbl@cite@choice{%
3756   \global\let\bibcite\bbl@bibcite
3757   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3758   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}}%
3759   \global\let\bbl@cite@choice\relax}
```

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

```
3760 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3761 \bbl@redefine\@bibitem#1{%
3762   \@safe@activestruel\org@@@bibitem{#1}\@safe@activesfalse}
3763 \else
3764   \let\org@nocite\nocite
3765   \let\org@@citex\@citex
3766   \let\org@babcite\babcite
3767   \let\org@@bibitem\@bibitem
3768 \fi
```

5.2. Layout

```
3769 \newcommand\BabelPatchSection[1]{%
3770   \@ifundefined{#1}{}{%
3771     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
3772     \@namedef{#1}{%
3773       \@ifstar{\bbl@presec@s{#1}}%
3774       {\@dblarg{\bbl@presec@x{#1}}}}%
3775 \def\bbl@presec@x#1[#2]#3{%
3776   \bbl@exp{%
3777     \\select@language@x{\bbl@main@language}%
3778     \\bbl@cs{sspre@#1}%
3779     \\bbl@cs{ss@#1}%
3780     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
3781     {\foreignlanguage{\language}{\unexpanded{#3}}}%
3782     \\select@language@x{\language}}%
3783 \def\bbl@presec@s#1#2{%
3784   \bbl@exp{%
3785     \\select@language@x{\bbl@main@language}%
3786     \\bbl@cs{sspre@#1}%
3787     \\bbl@cs{ss@#1}*%
3788     {\foreignlanguage{\language}{\unexpanded{#2}}}%
3789     \\xselect@language@x{\language}}%
3790 %
3791 \IfBabelLayout{sectioning}%
3792   {\BabelPatchSection{part}%
3793   \BabelPatchSection{chapter}%
3794   \BabelPatchSection{section}%
3795   \BabelPatchSection{subsection}%
3796   \BabelPatchSection{subsubsection}%
3797   \BabelPatchSection{paragraph}%
3798   \BabelPatchSection{subparagraph}}%
3799 \def\babel@toc#1{%
3800   \select@language@x{\bbl@main@language}}{}
3801 \IfBabelLayout{captions}%
3802   {\BabelPatchSection{caption}}{}
```

\BabelFootnote Footnotes.

```
3803 \bbl@trace{Footnotes}
3804 \def\bbl@footnote#1#2#3{%
3805   \@ifnextchar[%
3806     {\bbl@footnote@o{#1}{#2}{#3}}%
3807     {\bbl@footnote@x{#1}{#2}{#3}}%
3808 \long\def\bbl@footnote@x#1#2#3#4{%
3809   \bgroup
3810   \select@language@x{\bbl@main@language}%
3811   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%

```

```

3812 \egroup}
3813 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
3814 \bgroup
3815 \select@language@x{\bbl@main@language}%
3816 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3817 \egroup}
3818 \def\bbl@footnotetext#1#2#3{%
3819 \@ifnextchar[%
3820 {\bbl@footnotetext@o{#1}{#2}{#3}}%
3821 {\bbl@footnotetext@x{#1}{#2}{#3}}}
3822 \long\def\bbl@footnotetext@x#1#2#3#4{%
3823 \bgroup
3824 \select@language@x{\bbl@main@language}%
3825 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3826 \egroup}
3827 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
3828 \bgroup
3829 \select@language@x{\bbl@main@language}%
3830 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3831 \egroup}
3832 \def\BabelFootnote#1#2#3#4{%
3833 \ifx\bbl@fn@footnote\@undefined
3834 \let\bbl@fn@footnote\footnote
3835 \fi
3836 \ifx\bbl@fn@footnotetext\@undefined
3837 \let\bbl@fn@footnotetext\footnotetext
3838 \fi
3839 \bbl@ifblank{#2}%
3840 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3841 \namedef{\bbl@stripslash#1text}%
3842 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3843 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
3844 \namedef{\bbl@stripslash#1text}%
3845 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
3846 \IfBabelLayout{footnotes}%
3847 {\let\bbl@OL@footnote\footnote
3848 \BabelFootnote\footnote\languagename{}}}%
3849 \BabelFootnote\localfootnote\languagename{}}}%
3850 \BabelFootnote\mainfootnote{}}}%
3851 {}

```

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.

```

3852 \bbl@trace{Marks}
3853 \IfBabelLayout{sectioning}
3854 {\ifx\bbl@opt@headfoot\@nnil
3855 \g@addto@macro\resetactivechars{%
3856 \set@typeset@protect
3857 \expandafter\select@language@x\expandafter{\bbl@main@language}%
3858 \let\protect\noexpand
3859 \ifcase\bbl@bidimode\else % Only with bidi. See also above
3860 \edef\thepage{%
3861 \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3862 \fi}%
3863 \fi}
3864 {\ifbbl@single\else
3865 \bbl@ifunset{markright} \bbl@redefine\bbl@redefineroast

```

```

3866 \markright#1{%
3867   \bbl@ifblank{#1}%
3868   {\org@markright{}}%
3869   {\toks@{#1}%
3870   \bbl@exp{%
3871     \\org@markright{\\protect\\foreignlanguage{\language}%
3872     {\\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.)

```

3873 \ifx\@mkboth\markboth
3874   \def\bbl@tempc{\let\@mkboth\markboth}%
3875 \else
3876   \def\bbl@tempc{%
3877 \fi
3878 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3879 \markboth#1#2{%
3880   \protected@edef\bbl@tempb##1{%
3881     \protect\foreignlanguage
3882     {\language}{\protect\bbl@restore@actives##1}}%
3883   \bbl@ifblank{#1}%
3884     {\toks@{}}%
3885     {\toks@\expandafter{\bbl@tempb{#1}}}%
3886   \bbl@ifblank{#2}%
3887     {\@temptokena{}}%
3888     {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3889   \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}%
3890   \bbl@tempc
3891 \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.

```

3892 \bbl@trace{Preventing clashes with other packages}
3893 \ifx\org@ref@undefined\else
3894   \bbl@xin@{R}\bbl@opt@safe
3895   \ifin@
3896     \AtBeginDocument{%
3897       \@ifpackageloaded{ifthen}{%
3898         \bbl@redefine@longifthenelse#1#2#3{%

```

```

3899     \let\lbl@temp@pref\pageref
3900     \let\pageref\org@pageref
3901     \let\lbl@temp@ref\ref
3902     \let\ref\org@ref
3903     \@safe@activestruer
3904     \org@ifthenelse{#1}%
3905         {\let\pageref\lbl@temp@pref
3906          \let\ref\lbl@temp@ref
3907          \@safe@activesfalse
3908          #2}%
3909         {\let\pageref\lbl@temp@pref
3910          \let\ref\lbl@temp@ref
3911          \@safe@activesfalse
3912          #3}%
3913     }%
3914   }{}%
3915 }
3916 \fi

```

5.4.2. varioref

\@@vpageref

\vrefpagemum

\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 `\vrefpagemum`.

```

3917 \AtBeginDocument{%
3918   \ifpackageloaded{varioref}{%
3919     \bbl@redefine\@@vpageref#1[#2]#3{%
3920       \@safe@activestruer
3921       \org@@@vpageref{#1}[#2]{#3}%
3922       \@safe@activesfalse}%
3923     \bbl@redefine\vrefpagemum#1#2{%
3924       \@safe@activestruer
3925       \org@vrefpagemum{#1}{#2}%
3926       \@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.

```

3927   \expandafter\def\csname Ref \endcsname#1{%
3928     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3929   }{}%
3930 }
3931 \fi

```

5.4.3. hhlne

\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.

```

3932 \AtEndOfPackage{%
3933   \AtBeginDocument{%
3934     \ifpackageloaded{hhline}%
3935       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3936        \else
3937          \makeatletter
3938          \def\@currname{hhline}\input{hhline.sty}\makeatother

```

```

3939     \fi}%
3940     }}}

```

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

```

3941 \def\substitutefontfamily#1#2#3{%
3942   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3943   \immediate\write15{%
3944     \string\ProvidesFile{#1#2.fd}%
3945     [the\year/\two@digits{\the\month}/\two@digits{\the\day}
3946     \space generated font description file]^J
3947     \string\DeclareFontFamily{#1}{#2}{}}^J
3948     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^J
3949     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^J
3950     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^J
3951     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3952     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3953     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3954     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3955     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3956   }%
3957   \closeout15
3958 }
3959 \@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

```

3960 \bbl@trace{Encoding and fonts}
3961 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3962 \newcommand\BabelNonText{TS1,T3,TS3}
3963 \let\org@TeX\TeX
3964 \let\org@LaTeX\LaTeX
3965 \let\ensureascii@firstofone
3966 \let\asciencoding@empty
3967 \AtBeginDocument{%
3968   \def\@elt#1{,#1,}%
3969   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3970   \let\@elt\relax
3971   \let\bbl@tempb@empty
3972   \def\bbl@tempc{OT1}%
3973   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3974     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3975   \bbl@foreach\bbl@tempa{%
3976     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3977     \ifin@
3978       \def\bbl@tempb{#1}% Store last non-ascii
3979     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3980       \ifin@else
3981       \def\bbl@tempc{#1}% Store last ascii
3982       \fi
3983     \fi}%
3984   \ifx\bbl@tempb@empty\else
3985     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3986     \ifin@else

```



```

3987     \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3988     \fi
3989     \let\asciencoding\bbl@tempc
3990     \renewcommand\ensureasciil[1]{%
3991       {\fontencoding{\asciencoding}\selectfont#1}}%
3992     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3993     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3994     \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.

```

3995 \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.

```

3996 \AtBeginDocument{%
3997   \@ifpackageloaded{fontspec}%
3998   {\xdef\latinencoding{%
3999     \ifx\UTFencname@undefined
4000     EU\ifcase\bbl@engine\or2\or1\fi
4001     \else
4002     \UTFencname
4003     \fi}}%
4004   {\gdef\latinencoding{OT1}%
4005     \ifx\cf@encoding\bbl@t@one
4006     \xdef\latinencoding{\bbl@t@one}%
4007     \else
4008     \def\@elt#1{,#1,}%
4009     \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
4010     \let\@elt\relax
4011     \bbl@xin@{,T1,}\bbl@tempa
4012     \ifin@
4013     \xdef\latinencoding{\bbl@t@one}%
4014     \fi
4015     \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.

```

4016 \DeclareRobustCommand{\latintext}{%
4017   \fontencoding{\latinencoding}\selectfont
4018   \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.

```

4019 \ifx\@undefined\DeclareTextFontCommand
4020 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
4021 \else
4022 \DeclareTextFontCommand{\textlatin}{\latintext}
4023 \fi

```

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

```

4024 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.6. Basic bidi support

This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This `babel` module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I've also looked at `ARABI` (by Youssef Jabri), which is compatible with `babel`.

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like `rlbabel` did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `LuaTeX-ja` shows, vertical typesetting is possible, too.

```
4025 \bbl@trace{Loading basic (internal) bidi support}
4026 \ifodd\bbl@engine
4027 \else % Any xe+lua bidi
4028   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
4029     \bbl@error{bidi-only-lua}{\}\}\}%
4030     \let\bbl@beforeforeign\leavevmode
4031     \AtEndOfPackage{%
4032       \EnableBabelHook{babel-bidi}%
4033       \bbl@xebidipar}
4034   \fi\fi
4035   \def\bbl@loadxebidi#1{%
4036     \ifx\RTLfootnotetext\@undefined
4037       \AtEndOfPackage{%
4038         \EnableBabelHook{babel-bidi}%
4039         \ifx\fontspec\@undefined
4040           \usepackage{fontspec}% bidi needs fontspec
4041         \fi
4042         \usepackage#1{bidi}%
4043         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4044         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4045           \ifnum\@nameuse{\bbl@wdir\languagename}=\tw@ % 'AL' bidi
4046             \bbl@digitsdotdash % So ignore in 'R' bidi
4047           \fi}}%
4048     \fi}
4049   \ifnum\bbl@bidimode>200 % Any xe bidi=
4050     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4051       \bbl@tentative{bidi=bidi}
4052       \bbl@loadxebidi{}
4053     \or
4054       \bbl@loadxebidi{[rldocument]}
4055     \or
4056       \bbl@loadxebidi{}
4057     \fi
4058   \fi
4059 \fi
4060 \ifnum\bbl@bidimode=\@ne % bidi=default
4061   \let\bbl@beforeforeign\leavevmode
4062   \ifodd\bbl@engine % lua
4063     \newattribute\bbl@attr@dir
4064     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4065     \bbl@exp{\output{\bodydir\pagedir\the\output}}
```

```

4066 \fi
4067 \AtEndOfPackage{%
4068   \EnableBabelHook{babel-bidi}% pdf/luaxe
4069   \ifodd\bbbl@engine\else % pdf/xetex
4070     \bbbl@xetexbidipar
4071   \fi}
4072 \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.

```

4073 \bbbl@trace{Macros to switch the text direction}
4074 \def\bbbl@alscripts{%
4075   ,Arabic,Syriac,Thaana,Hanifi_Rohingya,Hanifi,Sogdian,}
4076 \def\bbbl@rscripts{%
4077   Adlam,Avestan,Chorasmian,Cypriot,Elymaic,Garay,%
4078   Hatran,Hebrew,Imperial_Aramaic,Inscriptional_Pahlavi,%
4079   Inscriptional_Parthian,Kharoshthi,Lydian,Mandaic,Manichaean,%
4080   Mende_Kikakui,Meroitic_Cursive,Meroitic_Hieroglyphs,Nabataean,%
4081   Nko,Old_Hungarian,Old_North_Arabian,Old_Sogdian,%
4082   Old_South_Arabian,Old_Turkic,Old_Uyghur,Palmyrene,Phoenician,%
4083   Psalter_Pahlavi,Samaritan,Yezidi,Mandaean,%
4084   Meroitic,N'Ko,Orkhon,Todhri}
4085 %
4086 \def\bbbl@provide@dirs#1{%
4087   \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
4088   \ifin@
4089     \global\bbbl@csarg\chardef{wdir@#1}\@ne
4090     \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
4091     \ifin@
4092       \global\bbbl@csarg\chardef{wdir@#1}\@two
4093     \fi
4094   \else
4095     \global\bbbl@csarg\chardef{wdir@#1}\@z
4096   \fi
4097   \ifodd\bbbl@engine
4098     \bbbl@csarg\ifcase{wdir@#1}%
4099     \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4100   \or
4101     \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4102   \or
4103     \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4104   \fi
4105 \fi}
4106 %
4107 \def\bbbl@switchdir{%
4108   \bbbl@ifunset{bbbl@sys@\languagename}{\bbbl@provide@sys{\languagename}}{%
4109     \bbbl@ifunset{bbbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}{%
4110       \bbbl@exp{\bbbl@setdirs\bbbl@c{l}{wdir}}
4111     \def\bbbl@setdirs#1{%
4112       \ifcase\bbbl@select@type
4113         \bbbl@bodydir{#1}%
4114         \bbbl@pardir{#1}% <- Must precede \bbbl@textdir
4115       \fi
4116       \bbbl@textdir{#1}}
4117     \ifnum\bbbl@bidimode>\@z
4118       \AddBabelHook{babel-bidi}{afterextras}{\bbbl@switchdir}
4119     \DisableBabelHook{babel-bidi}
4120 \fi}

```

Now the engine-dependent macros.

```

4121 \ifodd\bbbl@engine % luatex=1
4122 \else % pdftex=0, xetex=2
4123   \newcount\bbbl@dirlevel

```

```

4124 \chardef\bbL@thetextdir\z@
4125 \chardef\bbL@thepardir\z@
4126 \def\bbL@textdir#1{%
4127   \ifcase#1\relax
4128     \chardef\bbL@thetextdir\z@
4129     \@nameuse{setlatin}%
4130     \bbL@textdir@i\beginL\endL
4131   \else
4132     \chardef\bbL@thetextdir\@ne
4133     \@nameuse{setnonlatin}%
4134     \bbL@textdir@i\beginR\endR
4135   \fi}
4136 \def\bbL@textdir@i#1#2{%
4137   \ifhmode
4138     \ifnum\currentgrouplevel>\z@
4139       \ifnum\currentgrouplevel=\bbL@dirlevel
4140         \bbL@error{multiple-bidi}{\}\}%
4141         \bgroup\aftergroup#2\aftergroup\egroup
4142       \else
4143         \ifcase\currentgrouptype\or % 0 bottom
4144           \aftergroup#2% 1 simple {}
4145         \or
4146           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4147         \or
4148           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4149         \or\or\or % vbox vtop align
4150         \or
4151           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4152         \or\or\or\or\or\or % output math disc insert vcent mathchoice
4153         \or
4154           \aftergroup#2% 14 \begingroup
4155         \else
4156           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4157         \fi
4158       \fi
4159       \bbL@dirlevel\currentgrouplevel
4160     \fi
4161     #1%
4162   \fi}
4163 \def\bbL@pardir#1{\chardef\bbL@thepardir#1\relax}
4164 \let\bbL@bodydir@gobble
4165 \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).

```

4166 \def\bbL@xebidipar{%
4167   \let\bbL@xebidipar\relax
4168   \TeXeTstate\@ne
4169   \def\bbL@xeeverypar{%
4170     \ifcase\bbL@thepardir
4171       \ifcase\bbL@thetextdir\else\beginR\fi
4172     \else
4173       {\setbox\z@\lastbox\beginR\box\z@}%
4174     \fi}%
4175   \AddToHook{para/begin}{\bbL@xeeverypar}}
4176 \ifnum\bbL@bidimode>200 % Any xe bidi=
4177   \let\bbL@textdir@i@gobbletwo
4178   \let\bbL@xebidipar@empty
4179   \AddBabelHook{bidi}{foreign}{%
4180     \ifcase\bbL@thetextdir
4181       \BabelWrapText{\LR{##1}}%
4182     \else

```

```

4183     \BabelWrapText{\RL{##1}}%
4184     \fi}
4185     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4186     \fi
4187 \fi

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

4188 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4189 \AtBeginDocument{%
4190   \ifx\pdfstringdefDisableCommands\undefined\else
4191     \ifx\pdfstringdefDisableCommands\relax\else
4192       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4193     \fi
4194   \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`.

```

4195 \bbl@trace{Local Language Configuration}
4196 \ifx\loadlocalcfg\undefined
4197   \@ifpackagewith{babel}{noconfigs}%
4198   {\let\loadlocalcfg@gobble}%
4199   {\def\loadlocalcfg#1{%
4200     \InputIfFileExists{#1.cfg}%
4201     {\typeout{*****^J%
4202               * Local config file #1.cfg used^^J%
4203               *}}%
4204     \@empty}}
4205 \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).

```

4206 \bbl@trace{Language options}
4207 \def\BabelDefinitionFile#1#2#3{
4208 \let\bbl@afterlang\relax
4209 \let\BabelModifiers\relax
4210 \let\bbl@loaded@empty
4211 \def\bbl@load@language#1{%
4212   \InputIfFileExists{#1.ldf}%
4213   {\edef\bbl@loaded{\CurrentOption
4214     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
4215     \expandafter\let\expandafter\bbl@afterlang
4216     \csname\CurrentOption.ldf-h@k\endcsname
4217     \expandafter\let\expandafter\BabelModifiers
4218     \csname bbl@mod@\CurrentOption\endcsname
4219     \bbl@exp{\AtBeginDocument{%
4220       \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4221     {\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 as been set as metadata, read the info from the corresponding `ini` file and extract the babel name. Then added it as a package option at the end, so that it becomes the main language.

The behavior of a metatag with a global language option is not well defined, so if there is not a main option we set here explicitly.

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

```

4222 \ifx\GetDocumentProperties\undefined\else
4223 \let\bbl@beforeforeign\leavevmode
4224 \edef\bbl@metalang{\GetDocumentProperties{document/lang}}%
4225 \ifx\bbl@metalang\empty\else
4226 \begingroup
4227 \expandafter
4228 \bbl@bcplookup\bbl@metalang-\empty-\empty-\empty@@
4229 \ifx\bbl@bcp\relax
4230 \ifx\bbl@opt@main\@nnil
4231 \bbl@error{no-locale-for-meta}{\bbl@metalang}{}{}%
4232 \fi
4233 \else
4234 \bbl@read@ini{\bbl@bcp}\m@ne
4235 \xdef\bbl@language@opts{\bbl@language@opts,\language}%
4236 \ifx\bbl@opt@main\@nnil
4237 \global\let\bbl@opt@main\language
4238 \fi
4239 \bbl@info{Passing \language\space to babel.\\%
4240 This will be the main language except if\\%
4241 explicitly overridden with 'main='.\\%
4242 Reported}%
4243 \fi
4244 \endgroup
4245 \fi
4246 \fi
4247 \ifx\bbl@opt@config\@nnil
4248 \ifpackagewith{babel}{noconfigs}{}%
4249 {\InputIfFileExists{bblopts.cfg}%
4250 {\bbl@info{Configuration files are deprecated, as\\%
4251 they can break document portability.\\%
4252 Reported}%
4253 \typeout{*****^J%
4254 * Local config file bblopts.cfg used^J%
4255 *}}%
4256 {}}%
4257 \else
4258 \InputIfFileExists{\bbl@opt@config.cfg}%
4259 {\bbl@info{Configuration files are deprecated, as\\%
4260 they can break document portability.\\%
4261 Reported}%
4262 \typeout{*****^J%
4263 * Local config file \bbl@opt@config.cfg used^J%
4264 *}}%
4265 {\bbl@error{config-not-found}{}{}{}%
4266 \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`).

```

4267 \def\BabelBeforeIni#1#2{%
4268 \def\bbload@tempa{\@m}% <- Default if no \BDefFile
4269 \let\bbload@tempb\@empty
4270 #2%
4271 \edef\bbload@toLoad{%
4272 \ifx\bbload@toLoad\@empty\else\bbload@toLoad,\fi
4273 \bbload@toLoad@last}%
4274 \edef\bbload@toLoad@last{0/\bbload@tempa//\CurrentOption//#1/\bbload@tempb}}
4275 \def\BabelDefinitionFile#1#2#3{%
4276 \def\bbload@tempa{#1}\def\bbload@tempb{#2}%
4277 \@namedef{\bbload@preldf@\CurrentOption}{#3}%
4278 \endinput}%

```

For efficiency, first preprocess the class options to remove those with =, which are becoming increasingly frequent (no language should contain this character). Here we use the more robust macro to traverse a clist from the \TeX 3 layer.

```

4279 \def\bbload@tempf{,}
4280 \@nameuse{clist_map_inline:Nn}\@raw@classoptionslist{%
4281 \in@{=}{#1}%
4282 \ifin@ \else
4283 \edef\bbload@tempf{\bbload@tempf\zap@space#1 \@empty,}%
4284 \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.

```

4285 \let\bbload@toLoad\@empty
4286 \let\bbload@toLoad@last\@empty
4287 \let\bbload@unkopt\@gobble %% <- Ugly
4288 \edef\bbload@tempc{%
4289 \bbload@tempf,@@,\bbload@language@opts
4290 \ifx\bbload@opt@main\@nnil\else,\bbload@opt@main\fi}
4291 \let\BabelLocalesTentative\bbload@tempc
4292 %
4293 \bbload@foreach\bbload@tempc{%
4294 \in@{@@}{#1}% <- Ugly
4295 \ifin@
4296 \def\bbload@unkopt##1{%
4297 \DeclareOption{##1}{\bbload@error{unknown-package-option}{}}}%
4298 \else
4299 \def\CurrentOption{#1}%
4300 \bbload@xin@{/#1//}{\bbload@toLoad@last}% Collapse consecutive
4301 \ifin@ \else
4302 \lowercase{\InputIfFileExists{babel-#1.tex}}{%
4303 \IfFileExists{#1.ldf}%
4304 {\edef\bbload@toLoad{%
4305 \ifx\bbload@toLoad\@empty\else\bbload@toLoad,\fi
4306 \bbload@toLoad@last}%
4307 \edef\bbload@toLoad@last{0/0//\CurrentOption//und/#1}}%
4308 {\bbload@unkopt{#1}}}%
4309 \fi
4310 \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.

```

4311 \ifx\bbload@opt@main\@nnil
4312 \ifx\bbload@toLoad@last\@empty
4313 \def\bbload@toLoad@last{0/0//nil//und-x-nil/nil}
4314 \bbload@info{%
4315 You haven't specified a language as a class or package\%
4316 option. I'll load 'nil'. Reported}

```

```

4317 \fi
4318 \else
4319 \let\bbl@tempc\@empty
4320 \bbl@foreach\bbl@toload{%
4321 \bbl@xin@{/\bbl@opt@main//}{#1}%
4322 \ifin@ \else
4323 \bbl@add@list\bbl@tempc{#1}%
4324 \fi}
4325 \let\bbl@toload\bbl@tempc
4326 \fi
4327 \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'.

```

4328 \def\AfterBabelLanguage#1{%
4329 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4330 \NewHook{babel/presets}
4331 \UseHook{babel/presets}
4332 %
4333 \let\bbl@tempb\@empty
4334 \def\bbl@tempc#1/#2//#3//#4/#5\@@{%
4335 \count@ \z@
4336 \ifnum#2=\@m % if no \BabelDefinitionFile
4337 \ifnum#1=\z@ % not main. -- % if provide+!=, provide*!=
4338 \ifnum\bbl@ldfflag>\@ne\bbl@tempc 0/0//#3//#4/#3\@@
4339 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4340 \fi
4341 \else % 10 = main -- % if provide=!, provide*!=
4342 \ifodd\bbl@ldfflag\bbl@tempc 10/0//#3//#4/#3\@@
4343 \else\bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4344 \fi
4345 \fi
4346 \else
4347 \ifnum#1=\z@ % not main
4348 \ifnum\bbl@iniflag>\@ne \else % if ø, provide
4349 \ifcase#2\count@\@ne \else \ifcase\bbl@engine\count@\@ne \fi \fi
4350 \fi
4351 \else % 10 = main
4352 \ifodd\bbl@iniflag \else % if provide+, provide*
4353 \ifcase#2\count@\@ne \else \ifcase\bbl@engine\count@\@ne \fi \fi
4354 \fi
4355 \fi
4356 \bbl@tempd{#1}{#2}{#3}{#4}{#5}%
4357 \fi}

```

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

```

4358 \def\bbl@tempd#1#2#3#4#5{%
4359 \DeclareOption{#3}{}%
4360 \ifcase\count@
4361 \bbl@exp{\bbl@add{\bbl@tempb{%
4362 \bbl@nameuse{bbl@preini#3}%
4363 \bbl@ldfinit
4364 \def{\CurrentOption{#3}%
4365 \bbl@babelprovide[import=#4,\ifnum#1=\z@\else\bbl@opt@provide,main\fi]{#3}%
4366 \bbl@afterldf}}}%
4367 \else
4368 \bbl@add\bbl@tempb{%
4369 \def\CurrentOption{#3}%
4370 \let\localename\CurrentOption
4371 \let\languagename\localename
4372 \def\BabelIniTag{#4}%

```



```

4373     \@nameuse{bbl@preldf@#3}%
4374     \begingroup
4375         \bbl@id@assign
4376         \bbl@read@ini{\BabelIniTag}0%
4377     \endgroup
4378     \bbl@load@language{#5}%
4379 \fi}
4380 %
4381 \bbl@foreach\bbl@to\load{\bbl@tempc#1\@@}
4382 \bbl@tempb
4383 \DeclareOption*{}
4384 \ProcessOptions
4385 %
4386 \bbl@exp{%
4387     \\@AtBeginDocument{\bbl@usehooks@lang{/}{begindocument}{}}}%
4388 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}}
4389 </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`

```

4390 <{*kernel}
4391 \let\bbl@onlyswitch\@empty
4392 \input babel.def
4393 \let\bbl@onlyswitch\@undefined
4394 </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.

```

4395 <{*errors}
4396 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4397 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12
4398 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4399 \catcode`\@=11 \catcode`\^=7
4400 %
4401 \ifx\MessageBreak\@undefined
4402 \gdef\bbl@error@i#1#2{%
4403     \begingroup
4404         \newlinechar=`^^J
4405         \def\{^^J(babel) }%
4406         \errhelp{#2}\errmessage{\#1}%
4407     \endgroup}
4408 \else
4409 \gdef\bbl@error@i#1#2{%
4410     \begingroup
4411         \def\{\MessageBreak}%
4412         \PackageError{babel}{#1}{#2}%

```

```

4413   \endgroup}
4414 \fi
4415 \def\bbl@errmessage#1#2#3{%
4416   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4417     \bbl@error@i{#2}{#3}}
4418 % Implicit #2#3#4:
4419 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4420 %
4421 \bbl@errmessage{not-yet-available}
4422   {Not yet available}%
4423   {Find an armchair, sit down and wait}
4424 \bbl@errmessage{bad-package-option}%
4425   {Bad option '#1=#2'. Either you have misspelled the\\%
4426     key or there is a previous setting of '#1'. Valid\\%
4427     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4428     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4429   {See the manual for further details.}
4430 \bbl@errmessage{base-on-the-fly}
4431   {For a language to be defined on the fly 'base'\\%
4432     is not enough, and the whole package must be\\%
4433     loaded. Either delete the 'base' option or\\%
4434     request the languages explicitly}%
4435   {See the manual for further details.}
4436 \bbl@errmessage{undefined-language}
4437   {You haven't defined the language '#1' yet.\\%
4438     Perhaps you misspelled it or your installation\\%
4439     is not complete}%
4440   {Your command will be ignored, type <return> to proceed}
4441 \bbl@errmessage{invalid-ini-name}
4442   {'#1' not valid with the 'ini' mechanism.\\%
4443     I think you want '#2' instead. You may continue,\\%
4444     but you should fix the name. See the babel manual\\%
4445     for the available locales with 'provide'}%
4446   {See the manual for further details.}
4447 \bbl@errmessage{shorthand-is-off}
4448   {I can't declare a shorthand turned off (\string#2)}
4449   {Sorry, but you can't use shorthands which have been\\%
4450     turned off in the package options}
4451 \bbl@errmessage{not-a-shorthand}
4452   {The character '\string #1' should be made a shorthand character;\\%
4453     add the command \string\useshorthands\string{#1\string} to
4454     the preamble.\\%
4455     I will ignore your instruction}%
4456   {You may proceed, but expect unexpected results}
4457 \bbl@errmessage{not-a-shorthand-b}
4458   {I can't switch '\string#2' on or off--not a shorthand\\%
4459     This character is not a shorthand. Maybe you made\\%
4460     a typing mistake?}%
4461   {I will ignore your instruction.}
4462 \bbl@errmessage{unknown-attribute}
4463   {The attribute #2 is unknown for language #1.}%
4464   {Your command will be ignored, type <return> to proceed}
4465 \bbl@errmessage{missing-group}
4466   {Missing group for string \string#1}%
4467   {You must assign strings to some category, typically\\%
4468     captions or extras, but you set none}
4469 \bbl@errmessage{only-lua-xe}
4470   {This macro is available only in LuaLaTeX and XeLaTeX.}%
4471   {Consider switching to these engines.}
4472 \bbl@errmessage{only-lua}
4473   {This macro is available only in LuaLaTeX}%
4474   {Consider switching to that engine.}
4475 \bbl@errmessage{unknown-provide-key}

```

```

4476 {Unknown key '#1' in \string\babelprovide}%
4477 {See the manual for valid keys}%
4478 \bbl@errmessage{unknown-mapfont}
4479 {Option '\bbl@KVP@mapfont' unknown for\\%
4480 mapfont. Use 'direction'}%
4481 {See the manual for details.}
4482 \bbl@errmessage{no-ini-file}
4483 {There is no ini file for the requested language\\%
4484 (#1: \languagename). Perhaps you misspelled it or your\\%
4485 installation is not complete}%
4486 {Fix the name or reinstall babel.}
4487 \bbl@errmessage{digits-is-reserved}
4488 {The counter name 'digits' is reserved for mapping\\%
4489 decimal digits}%
4490 {Use another name.}
4491 \bbl@errmessage{limit-two-digits}
4492 {Currently two-digit years are restricted to the\\
4493 range 0-9999}%
4494 {There is little you can do. Sorry.}
4495 \bbl@errmessage{alphabetic-too-large}
4496 {Alphabetic numeral too large (#1)}%
4497 {Currently this is the limit.}
4498 \bbl@errmessage{no-ini-info}
4499 {I've found no info for the current locale.\\%
4500 The corresponding ini file has not been loaded\\%
4501 Perhaps it doesn't exist}%
4502 {See the manual for details.}
4503 \bbl@errmessage{unknown-ini-field}
4504 {Unknown field '#1' in \string\BCPdata.\\%
4505 Perhaps you misspelled it}%
4506 {See the manual for details.}
4507 \bbl@errmessage{unknown-locale-key}
4508 {Unknown key for locale '#2':\\%
4509 #3\\%
4510 \string#1 will be set to \string\relax}%
4511 {Perhaps you misspelled it.}%
4512 \bbl@errmessage{adjust-only-vertical}
4513 {Currently, #1 related features can be adjusted only\\%
4514 in the main vertical list}%
4515 {Maybe things change in the future, but this is what it is.}
4516 \bbl@errmessage{layout-only-vertical}
4517 {Currently, layout related features can be adjusted only\\%
4518 in vertical mode}%
4519 {Maybe things change in the future, but this is what it is.}
4520 \bbl@errmessage{bidi-only-lua}
4521 {The bidi method 'basic' is available only in\\%
4522 luatex. I'll continue with 'bidi=default', so\\%
4523 expect wrong results.\\%
4524 Suggested actions:\\%
4525 * If possible, switch to luatex, as xetex is not\\%
4526 recommend anymore.\\
4527 * If you can't, try 'bidi=bidi' with xetex.\\%
4528 * With pdftex, only 'bidi=default' is available.}%
4529 {See the manual for further details.}
4530 \bbl@errmessage{multiple-bidi}
4531 {Multiple bidi settings inside a group\\%
4532 I'll insert a new group, but expect wrong results.\\%
4533 Suggested action:\\%
4534 * Add a new group where appropriate.}
4535 {See the manual for further details.}
4536 \bbl@errmessage{unknown-package-option}
4537 {Unknown option '\CurrentOption'.\\%
4538 Suggested actions:\\%

```

```

4539 * Make sure you haven't misspelled it\\%
4540 * Check in the babel manual that it's supported\\%
4541 * If supported and it's a language, you may\\%
4542 \space\space need in some distributions a separate\\%
4543 \space\space installation\\%
4544 * If installed, check there isn't an old\\%
4545 \space\space version of the required files in your system\\%
4546 * If it's an unsupported language, create it with\\%
4547 \string\babelprovide (see the manual)}
4548 {Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4549 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4550 headfoot=, strings=, config=, hyphenmap=, or a language name.}
4551 \bbl@errmessage{config-not-found}
4552 {Local config file '\bbl@opt@config.cfg' not found.\\%
4553 Suggested actions:\\%
4554 * Make sure you haven't misspelled it in config=\\%
4555 * Check it exists and it's in the correct path}%
4556 {Perhaps you misspelled it.}
4557 \bbl@errmessage{late-after-babel}
4558 {Too late for \string\AfterBabelLanguage}%
4559 {Languages have been loaded, so I can do nothing}
4560 \bbl@errmessage{double-hyphens-class}
4561 {Double hyphens aren't allowed in \string\babelcharclass\\%
4562 because it's potentially ambiguous}%
4563 {See the manual for further info}
4564 \bbl@errmessage{unknown-interchar}
4565 {'#1' for '\languagename' cannot be enabled.\\%
4566 Maybe there is a typo}%
4567 {See the manual for further details.}
4568 \bbl@errmessage{unknown-interchar-b}
4569 {'#1' for '\languagename' cannot be disabled.\\%
4570 Maybe there is a typo}%
4571 {See the manual for further details.}
4572 \bbl@errmessage{charproperty-only-vertical}
4573 {\string\babelcharproperty\space can be used only in\\%
4574 vertical mode (preamble or between paragraphs)}%
4575 {See the manual for further info}
4576 \bbl@errmessage{unknown-char-property}
4577 {No property named '#2'. Allowed values are\\%
4578 direction (bc), mirror (bmg), and linebreak (lb)}%
4579 {See the manual for further info}
4580 \bbl@errmessage{bad-transform-option}
4581 {Bad option '#1' in a transform.\\%
4582 I'll ignore it but expect more errors}%
4583 {See the manual for further info.}
4584 \bbl@errmessage{font-conflict-transforms}
4585 {Transforms cannot be re-assigned to different\\%
4586 fonts. The conflict is in '\bbl@kv@label'.\\%
4587 Apply the same fonts or use a different label}%
4588 {See the manual for further details.}
4589 \bbl@errmessage{transform-not-available}
4590 {'#1' for '\languagename' cannot be enabled.\\%
4591 Maybe there is a typo or it's a font-dependent transform}%
4592 {See the manual for further details.}
4593 \bbl@errmessage{transform-not-available-b}
4594 {'#1' for '\languagename' cannot be disabled.\\%
4595 Maybe there is a typo or it's a font-dependent transform}%
4596 {See the manual for further details.}
4597 \bbl@errmessage{year-out-range}
4598 {Year out of range.\\%
4599 The allowed range is #1}%
4600 {See the manual for further details.}
4601 \bbl@errmessage{only-pdftex-lang}

```

```

4602 {The '#1' ldf style doesn't work with #2,\\%
4603 but you can use the ini locale instead.\\%
4604 Try adding 'provide=*' to the option list. You may\\%
4605 also want to set 'bidi=' to some value}%
4606 {See the manual for further details.}
4607 \bbl@errmessage{hyphenmins-args}
4608 {\string\babelhyphenmins\ accepts either the optional\\%
4609 argument or the star, but not both at the same time}%
4610 {See the manual for further details.}
4611 \bbl@errmessage{no-locale-for-meta}
4612 {There isn't currently a locale for the 'lang' requested\\%
4613 in the PDF metadata ('#1'). To fix it, you can\\%
4614 set explicitly a similar language (using the same\\%
4615 script) with the key main= when loading babel. If you\\%
4616 continue, I'll fallback to the 'nil' language, with\\%
4617 tag 'und' and script 'Latn', but expect a bad font\\%
4618 rendering with other scripts. You may also need set\\%
4619 explicitly captions and date, too}%
4620 {See the manual for further details.}
4621 </errors>
4622 <*patterns>

```

8. Loading hyphenation patterns

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

```

4623 <@Make sure ProvidesFile is defined@>
4624 \ProvidesFile{hyphen.cfg}[<@date@> v<@version@> Babel hyphens]
4625 \xdef\bbl@format{\jobname}
4626 \def\bbl@version{<@version@>}
4627 \def\bbl@date{<@date@>}
4628 \ifx\AtBeginDocument\undefined
4629 \def\@empty{}
4630 \fi
4631 <@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.

```

4632 \def\process@line#1#2 #3 #4 {%
4633 \ifx=#1%
4634 \process@synonym{#2}%
4635 \else
4636 \process@language{#1#2}{#3}{#4}%
4637 \fi
4638 \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.

```

4639 \toks@{}
4640 \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.

```

4641 \def\process@synonym#1{%
4642 \ifnum\last@language=\m@ne
4643 \toks@expandafter{\the\toks@\relax\process@synonym{#1}}%

```

```

4644 \else
4645   \expandafter\chardef\csname l@#1\endcsname\last@language
4646   \wlog{\string\l@#1=\string\language\the\last@language}%
4647   \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4648     \csname\language\name hyphenmins\endcsname
4649   \let\bb@elt\relax
4650   \edef\bb@languages{\bb@languages\bb@elt{#1}{\the\last@language}}}%
4651 \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 `\bb@get@enc` extracts the font encoding from the language name and stores it in `\bb@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

`\bb@languages` saves a snapshot of the loaded languages in the form `\bb@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.

```

4652 \def\process@language#1#2#3{%
4653   \expandafter\addlanguage\csname l@#1\endcsname
4654   \expandafter\language\csname l@#1\endcsname
4655   \edef\language#1{%
4656     \bb@hook@everylanguage{#1}%
4657     % > luatex
4658     \bb@get@enc#1::\@@@
4659   \begingroup
4660     \lefthyphenmin\m@ne
4661     \bb@hook@loadpatterns{#2}%
4662     % > luatex
4663     \ifnum\lefthyphenmin=\m@ne
4664     \else
4665       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4666         \the\lefthyphenmin\the\righthyphenmin}%
4667     \fi
4668   \endgroup
4669   \def\bb@tempa{#3}%
4670   \ifx\bb@tempa\@empty\else
4671     \bb@hook@loadexceptions{#3}%
4672     % > luatex
4673   \fi
4674   \let\bb@elt\relax
4675   \edef\bb@languages{\bb@languages\bb@elt{#1}{\the\language}{#2}{\bb@tempa}}%
4676   \ifnum\the\language=\z@

```

```

4678 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4679 \set@hyphenmins\tw@\thr@@\relax
4680 \else
4681 \expandafter\expandafter\expandafter\set@hyphenmins
4682 \csname #1hyphenmins\endcsname
4683 \fi
4684 \the\toks@
4685 \toks@{}%
4686 \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.

```

4687 \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.

```

4688 \def\bbl@hook@everylanguage#1{}
4689 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4690 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4691 \def\bbl@hook@loadkernel#1{%
4692 \def\addlanguage{\csname newlanguage\endcsname}%
4693 \def\adddialect##1##2{%
4694 \global\chardef##1##2\relax
4695 \wlog{\string##1 = a dialect from \string\language##2}}%
4696 \def\iflanguage##1{%
4697 \expandafter\ifx\csname l@##1\endcsname\relax
4698 \@nolanerr{##1}%
4699 \else
4700 \ifnum\csname l@##1\endcsname=\language
4701 \expandafter\expandafter\expandafter\@firstoftwo
4702 \else
4703 \expandafter\expandafter\expandafter\@secondoftwo
4704 \fi
4705 \fi}%
4706 \def\providehyphenmins##1##2{%
4707 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4708 \@namedef{##1hyphenmins}{##2}%
4709 \fi}%
4710 \def\set@hyphenmins##1##2{%
4711 \lefthyphenmin##1\relax
4712 \righthyphenmin##2\relax}%
4713 \def\selectlanguage{%
4714 \errhelp{Selecting a language requires a package supporting it}%
4715 \errmessage{No multilingual package has been loaded}}%
4716 \let\foreignlanguage\selectlanguage
4717 \let\otherlanguage\selectlanguage
4718 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4719 \def\bbl@usehooks##1##2{%
4720 \def\setlocale{%
4721 \errhelp{Find an armchair, sit down and wait}%
4722 \errmessage{(babel) Not yet available}}%
4723 \let\uselocale\setlocale
4724 \let\locale\setlocale
4725 \let\selectlocale\setlocale
4726 \let\localename\setlocale
4727 \let\textlocale\setlocale
4728 \let\textlanguage\setlocale
4729 \let\languagetext\setlocale}
4730 \begingroup
4731 \def\AddBabelHook#1#2{%
4732 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax

```

```

4733     \def\next{\toks1}%
4734     \else
4735     \def\next{\expandafter\gdef\cname bbl@hook@#2\endcsname###1}%
4736     \fi
4737     \next}
4738 \ifx\directlua\@undefined
4739 \ifx\XeTeXinputencoding\@undefined\else
4740     \input xebabel.def
4741     \fi
4742 \else
4743     \input luababel.def
4744     \fi
4745 \openin1 = babel-\bbl@format.cfg
4746 \ifeof1
4747 \else
4748     \input babel-\bbl@format.cfg\relax
4749     \fi
4750 \closein1
4751 \endgroup
4752 \bbl@hook@loadkernel{switch.def}

```

readconfigfile The configuration file can now be opened for reading.

```

4753 \openin1 = language.dat

```

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

```

4754 \def\languagename{english}%
4755 \ifeof1
4756 \message{I couldn't find the file language.dat,\space
4757           I will try the file hyphen.tex}
4758 \input hyphen.tex\relax
4759 \chardef\l@english\z@
4760 \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`.

```

4761 \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.

```

4762 \loop
4763 \endlinechar@m@ne
4764 \read1 to \bbl@line
4765 \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.

```

4766 \if T\ifeof1\fi T\relax
4767 \ifx\bbl@line\@empty\else
4768     \edef\bbl@line{\bbl@line\space\space\space}%
4769     \expandafter\process@line\bbl@line\relax
4770     \fi
4771 \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.

```

4772 \begingroup
4773 \def\bbl@elt#1#2#3#4{%
4774     \global\language=#2\relax

```



```

4775 \gdef\languagename{#1}%
4776 \def\bbl@elt##1##2##3##4{}}%
4777 \bbl@languages
4778 \endgroup
4779 \fi
4780 \closein1

```

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

```

4781 \if/\the\toks@/\else
4782 \errhelp{language.dat loads no language, only synonyms}
4783 \errmessage{Orphan language synonym}
4784 \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.

```

4785 \let\bbl@line\@undefined
4786 \let\process@line\@undefined
4787 \let\process@synonym\@undefined
4788 \let\process@language\@undefined
4789 \let\bbl@get@enc\@undefined
4790 \let\bbl@hyph@enc\@undefined
4791 \let\bbl@tempa\@undefined
4792 \let\bbl@hook@loadkernel\@undefined
4793 \let\bbl@hook@everylanguage\@undefined
4794 \let\bbl@hook@loadpatterns\@undefined
4795 \let\bbl@hook@loadexceptions\@undefined
4796 </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`).

```

4797 <<*More package options>> ≡
4798 \chardef\bbl@bidimode\z@
4799 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4800 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4801 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4802 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4803 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4804 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4805 <</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`.

```

4806 <<*Font selection>> ≡
4807 \bbl@trace{Font handling with fontspec}
4808 \AddBabelHook{babel - fontspec}{afterextras}{\bbl@switchfont}
4809 \AddBabelHook{babel - fontspec}{beforestart}{\bbl@cckckstdfont}
4810 \DisableBabelHook{babel - fontspec}
4811 \@onlypreamble\babelfont
4812 \ifx\NewDocumentCommand\@undefined\else % Not plain
4813 \NewDocumentCommand\babelfont{0}{m0}{m0}{\%
4814 \bbl@bblfont@o[#1]{#2}[#3,#5]{#4}}
4815 \fi
4816 \newcommand\bbl@bblfont@o[2][\% 1=langs/scripts 2=fam
4817 \ifx\fontspec\@undefined
4818 \usepackage{fontspec}%

```

```

4819 \fi
4820 \EnableBabelHook{babel-fontspec}%
4821 \edef\bbbl@tempa{#1}%
4822 \def\bbbl@tempb{#2}% Used by \bbbl@bblfont
4823 \bbbl@bblfont}
4824 \newcommand\bbbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4825 \bbbl@ifunset{\bbbl@tempb family}%
4826 {\bbbl@providefam{\bbbl@tempb}}%
4827 {}%
4828 % For the default font, just in case:
4829 \bbbl@ifunset{\bbbl@sys@\languagename}{\bbbl@provide@sys{\languagename}}{%
4830 \expandafter\bbbl@ifblank\expandafter{\bbbl@tempa}%
4831 {\bbbl@csarg\edef{\bbbl@tempb dflt@}{<#1>{#2}}% save bbl@rmdflt@
4832 \bbbl@exp{%
4833 \let<\bbbl@tempb dflt@\languagename><\bbbl@tempb dflt@>%
4834 \\\bbbl@font@set<\bbbl@tempb dflt@\languagename>%
4835 \<\bbbl@tempb default><\bbbl@tempb family>}}%
4836 {\bbbl@foreach\bbbl@tempa{% i.e., bbl@rmdflt@lang / *scrt
4837 \bbbl@csarg\def{\bbbl@tempb dflt@##1}{<#1>{#2}}}}%

```

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

```

4838 \def\bbbl@providefam#1{%
4839 \bbbl@exp{%
4840 \\\newcommand<#1default>{}% Just define it
4841 \\\bbbl@add@list\\bbbl@font@fams{#1}%
4842 \\\NewHook{#1family}%
4843 \\\DeclareRobustCommand<#1family>{%
4844 \\\not@math@alphabet<#1family>\relax
4845 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4846 \\\fontfamily<#1default>%
4847 \\\UseHook{#1family}%
4848 \\\selectfont}%
4849 \\\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.

```

4850 \def\bbbl@nostdfont#1{%
4851 \bbbl@once{nostdfam-\f@family}%
4852 {\bbbl@infowarn{The current font is not a babel standard family:\\%
4853 #1%
4854 \fontname\font\\%
4855 There is nothing intrinsically wrong, and you can\\%,
4856 ignore this message altogether if you do not need\\%
4857 this font. If they are used in the document, be aware\\%
4858 'babel' will not set Script and Language for it, so\\%
4859 you may consider defining a new family with \string\babelfont.\\%
4860 See the manual for further details about \string\babelfont.
4861 Reported}}%
4862 {}}%
4863 \gdef\bbbl@switchfont{%
4864 \bbbl@ifunset{\bbbl@sys@\languagename}{\bbbl@provide@sys{\languagename}}{%
4865 \bbbl@exp{% e.g., Arabic -> arabic
4866 \\\lowercase{\edef\\bbbl@tempa{\bbbl@c{l{sname}}}}}%
4867 \bbbl@foreach\bbbl@font@fams{%
4868 \bbbl@ifunset{\bbbl@##1dflt@\languagename}% (1) language?
4869 {\bbbl@ifunset{\bbbl@##1dflt@*\bbbl@tempa}% (2) from script?
4870 {\bbbl@ifunset{\bbbl@##1dflt@}% 2=F - (3) from generic?
4871 {}% 123=F - nothing!
4872 {\bbbl@exp{% 3=T - from generic
4873 \global\let<\bbbl@##1dflt@\languagename>%
4874 <\bbbl@##1dflt@>}}}%
4875 {\bbbl@exp{% 2=T - from script
4876 \global\let<\bbbl@##1dflt@\languagename>%
4877 <\bbbl@##1dflt@*\bbbl@tempa>}}}%

```

```

4878     {}}%                                l=T - language, already defined
4879 \def\bbl@tempa{\bbl@nostdfont{}}%
4880 \bbl@foreach\bbl@font@fams{%    don't gather with prev for
4881   \bbl@ifunset{\bbl@##1dflt@\languagename}%
4882   {\bbl@cs{famrst@##1}%
4883    \global\bbl@csarg\let{famrst@##1}\relax}%
4884   {\bbl@exp{% order is relevant.
4885     \\bbl@add\\originalTeX{%
4886     \\bbl@font@rst{\bbl@cl{##1dflt}}%
4887     \<##1default>\<##1family>{##1}}%
4888     \\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4889     \<##1default>\<##1family>}}}%
4890 \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`.

```

4891 \ifx\f@family\undefined\else    % if latex
4892   \ifcase\bbl@engine              % if pdftex
4893     \let\bbl@cckstfont\relax
4894   \else
4895     \def\bbl@cckstfont{%
4896       \begingroup
4897       \global\let\bbl@cckstfont\relax
4898       \let\bbl@tempa\empty
4899       \bbl@foreach\bbl@font@fams{%
4900         \bbl@ifunset{\bbl@##1dflt@}%
4901         {\@nameuse{##1family}%
4902          \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4903          \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \f@family\\}%
4904           \space\space\fontname\font\\}}%
4905          \bbl@csarg\xdef{##1dflt@}{\f@family}%
4906          \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4907         {}}%
4908     \ifx\bbl@tempa\empty\else
4909       \bbl@infowarn{The following font families will use the default\\%
4910         settings for all or some languages:\\%
4911         \bbl@tempa
4912         There is nothing intrinsically wrong with it, but\\%
4913         'babel' will no set Script and Language, which could\\%
4914         be relevant in some languages. If your document uses\\%
4915         these families, consider redefining them with \string\babelfont.\\%
4916         Reported}%
4917     \fi
4918   \endgroup}
4919 \fi
4920 \fi

```

Now the macros defining the font with `fontspec`.

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

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

```

4921 \def\bbl@font@set#1#2#3{% e.g., \bbl@rmdflt@lang \rmdefault \rmfamily
4922   \bbl@xin@{<>}{#1}%
4923   \ifin@
4924     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4925   \fi
4926   \bbl@exp{% 'Unprotected' macros return prev values
4927     \def\\#2{#1}% e.g., \rmdefault{\bbl@rmdflt@lang}

```

```

4928   \\bbl@ifsamestring{#2}{\f@family}%
4929   {\#3%
4930   \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4931   \let\\bbl@tempa\relax}%
4932   {}}}

```

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).

```

4933 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4934   \let\bbl@tempe\bbl@mapselect
4935   \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4936   \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}}}%
4937   \let\bbl@mapselect\relax
4938   \let\bbl@tempfam#4%          e.g., '\rmfamily', to be restored below
4939   \let#4@empty %              Make sure \renewfontfamily is valid
4940   \bbl@set@renderer
4941   \bbl@exp{%
4942     \let\\bbl@temp@pfam<\bbl@stripslash#4\space> e.g., '\rmfamily '
4943     \<keys_if_exist:nf>{fontspec-opentype}{Script/\bbl@cl{sname}}}%
4944     {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4945     \<keys_if_exist:nf>{fontspec-opentype}{Language/\bbl@cl{lname}}}%
4946     {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4947     \\renewfontfamily\\#4%
4948     [\bbl@cl{lsys},% xetex removes unknown features :-(
4949     \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4950     #2]}{#3}% i.e., \bbl@exp{.}{#3}
4951   \bbl@unset@renderer
4952   \begingroup
4953     #4%
4954     \xdef#1{\f@family}%      e.g., \bbl@rmdflt@lang{FreeSerif(0)}
4955   \endgroup
4956   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4957   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4958   \ifin@
4959     \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4960   \fi
4961   \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4962   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4963   \ifin@
4964     \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4965   \fi
4966   \let#4\bbl@tempfam
4967   \bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam
4968   \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.

```

4969 \def\bbl@font@rst#1#2#3#4{%
4970   \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`.

```

4971 \def\bbl@font@fams{rm,sf,tt}
4972 <</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.

```
4973 (*xetex)
4974 \def\BabelStringsDefault{unicode}
4975 \let\xebbl@stop\relax
4976 \AddBabelHook{xetex}{encodedcommands}{%
4977   \def\bbl@tempa{#1}%
4978   \ifx\bbl@tempa\@empty
4979     \XeTeXinputencoding"bytes"%
4980   \else
4981     \XeTeXinputencoding"#1"%
4982   \fi
4983   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4984 \AddBabelHook{xetex}{stopcommands}{%
4985   \xebbl@stop
4986   \let\xebbl@stop\relax}
4987 \def\bbl@input@classes{% Used in CJK intraspaces
4988   \input{load-unicode-xetex-classes.tex}%
4989   \let\bbl@input@classes\relax}
4990 \def\bbl@intraspace#1 #2 #3\@{%
4991   \bbl@csarg\gdef{xeisp@\language}%
4992     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4993 \def\bbl@intrapenalty#1\@{%
4994   \bbl@csarg\gdef{xeipn@\language}%
4995     {\XeTeXlinebreakpenalty #1\relax}}
4996 \def\bbl@provide@intraspace{%
4997   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4998   \ifin@else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4999   \ifin@
5000     \bbl@ifunset{bbl@intsp@\language}{}%
5001     {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
5002       \ifx\bbl@KVP@intraspace\@nnil
5003         \bbl@exp{%
5004           \\bbl@intraspace\bbl@cl{intsp}\@}%
5005         \fi
5006         \ifx\bbl@KVP@intrapenalty\@nnil
5007           \bbl@intrapenalty0\@
5008         \fi
5009         \fi
5010         \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
5011           \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
5012         \fi
5013         \ifx\bbl@KVP@intrapenalty\@nnil\else
5014           \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5015         \fi
5016         \bbl@exp{%
5017           \\bbl@add\<extras\language>%
5018           \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
5019           \<bbl@xeisp@\language>%
5020           \<bbl@xeipn@\language>%
5021           \\bbl@tglobal\<extras\language>%
5022           \\bbl@add\<noextras\language>%
5023           \XeTeXlinebreaklocale ""}%
5024           \\bbl@tglobal\<noextras\language>%
5025         \ifx\bbl@ispacesize\undefined
5026           \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
5027         \ifx\AtBeginDocument\@notprerr
5028           \expandafter\@secondoftwo % to execute right now
5029         \fi
5030         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
5031       \fi}%
5032   \fi}
5033 \ifx\DisableBabelHook\undefined\endinput\fi
5034 \let\bbl@set@renderer\relax
```

```

5035 \let\bbl@unset@renderer\relax
5036 <@Font selection>
5037 \def\bbl@provide@extra#1{}

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

5038 \def\bbl@xenoxyph@d{%
5039 \bbl@ifset{bbl@prehc@}\languagename}%
5040 {\ifnum\hyphenchar\font=\defaulthyphenchar
5041 \iffontchar\font\bbl@cl{prehc}\relax
5042 \hyphenchar\font\bbl@cl{prehc}\relax
5043 \else\iffontchar\font"200B
5044 \hyphenchar\font"200B
5045 \else
5046 \bbl@warning
5047 {Neither 0 nor ZERO WIDTH SPACE are available\\%
5048 in the current font, and therefore the hyphen\\%
5049 will be printed. Try changing the fontspec's\\%
5050 'HyphenChar' to another value, but be aware\\%
5051 this setting is not safe (see the manual).\\%
5052 Reported}%
5053 \hyphenchar\font\defaulthyphenchar
5054 \fi\fi
5055 \fi}%
5056 {\hyphenchar\font\defaulthyphenchar}}

```

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.

```

5057 \ifnum\xe@alloc@intercharclass<\thr@@
5058 \xe@alloc@intercharclass\thr@@
5059 \fi
5060 \chardef\bbl@xe@class@default@=\z@
5061 \chardef\bbl@xe@class@cjkideogram@=\@ne
5062 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
5063 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
5064 \chardef\bbl@xe@class@boundary@=4095
5065 \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@usingxeclass, defined below. The standard mechanism based on \originalTeX to save, set and restore values is used. \count@ stores the previous char to be set, except at the beginning (0) and after \bbl@upto, which is the previous char negated, as a flag to mark a range.

```

5066 \AddBabelHook{babel-interchar}{beforeextras}{%
5067 \@nameuse{bbl@xechars@\languagename}}
5068 \DisableBabelHook{babel-interchar}
5069 \protected\def\bbl@charclass#1{%
5070 \ifnum\count@<\z@
5071 \count@-\count@
5072 \loop
5073 \bbl@exp{%
5074 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
5075 \XeTeXcharclass\count@ \bbl@tempc
5076 \ifnum\count@<`#1\relax
5077 \advance\count@\@ne
5078 \repeat
5079 \else
5080 \babel@savevariable{\XeTeXcharclass`#1}%
5081 \XeTeXcharclass`#1 \bbl@tempc
5082 \fi
5083 \count@`#1\relax}

```

Now the two user macros. Char classes are declared implicitly, and then the macro to be executed at the babel-interchar hook is created. The list of chars to be handled by the hook defined above

has internally the form `\bbl@usingxeclasse\bbl@xeclasse@punct@english\bbl@charclass{.}`
`\bbl@charclass{,}` (etc.), where `\bbl@usingxeclasse` stores the class to be applied to the
subsequent characters. The `\ifcat` part deals with the alternative way to enter characters as macros
(e.g., `\`). As a special case, hyphens are stored as `\bbl@upto`, to deal with ranges.

```

5084 \newcommand\bbl@ifinterchar[1]{%
5085   \let\bbl@tempa@gobble      % Assume to ignore
5086   \edef\bbl@tempb{\zap@space#1 \@empty}%
5087   \ifx\bbl@KVP@interchar\@nnil\else
5088     \bbl@replace\bbl@KVP@interchar{ }{,}%
5089     \bbl@foreach\bbl@tempb{%
5090       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5091       \ifin@
5092         \let\bbl@tempa@firstofone
5093       \fi}%
5094   \fi
5095   \bbl@tempa}
5096 \newcommand\IfBabelIntercharT[2]{%
5097   \bbl@csarg\bbl@add{\bbl@icsave@CurrentOption}{\bbl@ifinterchar{#1}{#2}}}%
5098 \newcommand\babelcharclass[3]{%
5099   \EnableBabelHook{babel-interchar}%
5100   \bbl@csarg\newXeTeXintercharclass{xeclasse@#2@#1}%
5101   \def\bbl@tempb##1{%
5102     \ifx##1@empty\else
5103       \ifx##1-%
5104         \bbl@upto
5105       \else
5106         \bbl@charclass{%
5107           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5108         \fi
5109         \expandafter\bbl@tempb
5110       \fi}%
5111   \bbl@ifunset{\bbl@xechars@#1}%
5112   {\toks@{%
5113     \babel@savevariable\XeTeXinterchartokenstate
5114     \XeTeXinterchartokenstate\@ne
5115   }}%
5116   {\toks@\expandafter\expandafter\expandafter{%
5117     \csname bbl@xechars@#1\endcsname}}%
5118   \bbl@csarg\edef{xechars@#1}{%
5119     \the\toks@
5120     \bbl@usingxeclasse\csname bbl@xeclasse@#2@#1\endcsname
5121     \bbl@tempb#3@empty}}
5122 \protected\def\bbl@usingxeclasse#1{\count@>\z@ \let\bbl@tempc#1}
5123 \protected\def\bbl@upto{%
5124   \ifnum\count@>\z@
5125     \advance\count@\@ne
5126     \count@-\count@
5127   \else\ifnum\count@=\z@
5128     \bbl@charclass{-}%
5129   \else
5130     \bbl@error{double-hyphens-class}{ }{}%
5131   \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>`.

```

5132 \def\bbl@ignoreinterchar{%
5133   \ifnum\language=\l@nohyphenation
5134     \expandafter@gobble
5135   \else
5136     \expandafter@firstofone
5137   \fi}
5138 \newcommand\babelinterchar[5][ ]{%

```

```

5139 \let\bbk@kv@label\@empty
5140 \bbk@forkv{#1}{\bbk@csarg\edef{kv@##1}{##2}}%
5141 \namedef{\zap@space bbl@xeinter@\bbk@kv@label @#3@#4@#2 \@empty}%
5142 {\bbk@ignoreinterchar{#5}}%
5143 \bbk@csarg\let{ic@\bbk@kv@label @#2}\@firstofone
5144 \bbk@exp{\bbk@for\bbk@tempa{\zap@space#3 \@empty}}{%
5145 \bbk@exp{\bbk@for\bbk@tempb{\zap@space#4 \@empty}}{%
5146 \XeTeXinterchartoks
5147 \@nameuse{bbl@xeiclass@\bbk@tempa @%
5148 \bbk@ifunset{bbl@xeiclass@\bbk@tempa @#2}{#2}} %
5149 \@nameuse{bbl@xeiclass@\bbk@tempb @%
5150 \bbk@ifunset{bbl@xeiclass@\bbk@tempb @#2}{#2}} %
5151 = \expandafter{%
5152 \csname bbl@ic@\bbk@kv@label @#2\expandafter\endcsname
5153 \csname\zap@space bbl@xeinter@\bbk@kv@label
5154 @#3@#4@#2 \@empty\endcsname}}}}
5155 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5156 \bbk@ifunset{bbl@ic@#1@languagename}%
5157 {\bbk@error{unknown-interchar}{#1}{}}%
5158 {\bbk@csarg\let{ic@#1@languagename}\@firstofone}}
5159 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5160 \bbk@ifunset{bbl@ic@#1@languagename}%
5161 {\bbk@error{unknown-interchar-b}{#1}{}}%
5162 {\bbk@csarg\let{ic@#1@languagename}\@gobble}}
5163 </xetex>

```

10.3. Layout

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

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T_EX 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.

```

5164 (*xetex | texxet)
5165 \providecommand\bbl@provide@intraspace{}
5166 \bbl@trace{Redefinitions for bidi layout}

Finish here if there in no layout.

5167 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5168 \IfBabelLayout{nopars}
5169 {}
5170 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
5171 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5172 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5173 \ifnum\bbl@bidimode>\z@
5174 \IfBabelLayout{pars}
5175 {\def\@hangfrom#1{%
5176 \setbox\@tempboxa\hbox{#1}}%
5177 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5178 \noindent\box\@tempboxa}
5179 \def\raggedright{%
5180 \let\@centercr
5181 \bbl@startskip\z@skip
5182 \@rightskip\@flushglue
5183 \bbl@endskip\@rightskip
5184 \parindent\z@
5185 \parfillskip\bbl@startskip}
5186 \def\raggedleft{%
5187 \let\@centercr
5188 \bbl@startskip\@flushglue
5189 \bbl@endskip\z@skip

```



```

5190     \parindent\z@
5191     \parfillskip\bbbl@endskip}}
5192 {}
5193 \fi
5194 \IfBabelLayout{lists}
5195   {\bbbl@sreplace\list
5196     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbbl@listleftmargin}%
5197     \def\bbbl@listleftmargin{%
5198       \ifcase\bbbl@thepardir\leftmargin\else\rightmargin\fi}%
5199     \ifcase\bbbl@engine
5200       \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
5201       \def\p@enumiii{\p@enumii}\theenumii{}%
5202     \fi
5203     \bbbl@sreplace\@verbatim
5204       {\leftskip\@totalleftmargin}%
5205       {\bbbl@startskip\textwidth
5206         \advance\bbbl@startskip-\linewidth}%
5207     \bbbl@sreplace\@verbatim
5208       {\rightskip\z@skip}%
5209       {\bbbl@endskip\z@skip}}%
5210 {}
5211 \IfBabelLayout{contents}
5212   {\bbbl@sreplace\@dottedtocline{\leftskip}{\bbbl@startskip}%
5213     \bbbl@sreplace\@dottedtocline{\rightskip}{\bbbl@endskip}}
5214 {}
5215 \IfBabelLayout{columns}
5216   {\bbbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbbl@outputbox}%
5217     \def\bbbl@outputbox#1{%
5218       \hb@xt@\textwidth{%
5219         \hskip\columnwidth
5220         \hfil
5221         {\normalcolor\vrule \@width\columnseprule}%
5222         \hfil
5223         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5224         \hskip-\textwidth
5225         \hb@xt@\columnwidth{\box\@outputbox \hss}%
5226         \hskip\columnsep
5227         \hskip\columnwidth}}}%
5228   {}

```

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

```

5229 \IfBabelLayout{counters*}%
5230   {\bbbl@add\bbbl@opt@layout{.counters.}%
5231     \AddToHook{shipout/before}{%
5232       \let\bbbl@tempa\babelsublr
5233       \let\babelsublr\@firstofone
5234       \let\bbbl@save@thepage\thepage
5235       \protected@edef\thepage{\thepage}%
5236       \let\babelsublr\bbbl@tempa}%
5237     \AddToHook{shipout/after}{%
5238       \let\thepage\bbbl@save@thepage}}{}
5239 \IfBabelLayout{counters}%
5240   {\let\bbbl@latinarabic=\@arabic
5241     \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
5242     \let\bbbl@asciroman=\@roman
5243     \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
5244     \let\bbbl@asciiRoman=\@Roman
5245     \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}}{}
5246 \fi % end if layout
5247 </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).

```
5248 (*texxet)
5249 \def\bbl@provide@extra#1{%
5250   % == auto-select encoding ==
5251   \ifx\bbl@encoding@select@off\@empty\else
5252     \bbl@ifunset\bbl@encoding@#1{%
5253       {\def\elt##1{,##1,}%
5254        \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5255        \count@z@
5256        \bbl@foreach\bbl@tempe{%
5257          \def\bbl@tempd{##1}% Save last declared
5258          \advance\count@\@ne}%
5259        \ifnum\count@>\@ne % (1)
5260          \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5261          \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5262          \bbl@replace\bbl@tempa{ }{,}%
5263          \global\bbl@csarg\let{encoding@#1}\@empty
5264          \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5265          \ifin\@else % if main encoding included in ini, do nothing
5266            \let\bbl@tempb\relax
5267            \bbl@foreach\bbl@tempa{%
5268              \ifx\bbl@tempb\relax
5269                \bbl@xin@{,##1,}{,\bbl@tempe,}%
5270                \ifin\def\bbl@tempb{##1}\fi
5271              \fi}%
5272            \ifx\bbl@tempb\relax\else
5273              \bbl@exp{%
5274                \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>}%
5275              \gdef\<bbl@encoding@#1>{%
5276                \\babel@save\\f@encoding
5277                \\bbl@add\\originalTeX{\\selectfont}%
5278                \\fontencoding{\bbl@tempb}%
5279                \\selectfont}}%
5280              \fi
5281            \fi
5282          \fi}%
5283        }%
5284      \fi}
5285 (</texxet)
```

10.5. LuaTeX

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

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

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

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

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

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

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

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

```

5286 (*luatex)
5287 \directlua{ Babel = Babel or {} } % DL2
5288 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5289 \bbl@trace{Read language.dat}
5290 \ifx\bbl@readstream\undefined
5291 \csname newread\endcsname\bbl@readstream
5292 \fi
5293 \begingroup
5294 \toks@{}
5295 \count@% \z@ % 0=start, 1=0th, 2=normal
5296 \def\bbl@process@line#1#2 #3 #4 {%
5297 \ifx=#1%
5298 \bbl@process@synonym{#2}%
5299 \else
5300 \bbl@process@language{#1#2}{#3}{#4}%
5301 \fi
5302 \ignorespaces}
5303 \def\bbl@manylang{%
5304 \ifnum\bbl@last>\@ne
5305 \bbl@info{Non-standard hyphenation setup}%
5306 \fi
5307 \let\bbl@manylang\relax}
5308 \def\bbl@process@language#1#2#3{%
5309 \ifcase\count@
5310 \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5311 \or
5312 \count@\tw@
5313 \fi
5314 \ifnum\count@=\tw@
5315 \expandafter\addlanguage\csname l@#1\endcsname
5316 \language\allocationnumber
5317 \chardef\bbl@last\allocationnumber
5318 \bbl@manylang
5319 \let\bbl@elt\relax
5320 \xdef\bbl@languages{%
5321 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5322 \fi
5323 \the\toks@
5324 \toks@{}}
5325 \def\bbl@process@synonym@aux#1#2{%
5326 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5327 \let\bbl@elt\relax
5328 \xdef\bbl@languages{%
5329 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5330 \def\bbl@process@synonym#1{%
5331 \ifcase\count@
5332 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5333 \or

```

```

5334     \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
5335 \else
5336     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5337 \fi}
5338 \ifx\bbl@languages\undefined % Just a (sensible?) guess
5339 \chardef\l@english\z@
5340 \chardef\l@USenglish\z@
5341 \chardef\bbl@last\z@
5342 \global\@namedef{bbl@hyphendata@0}{{\hyphen.tex}}
5343 \gdef\bbl@languages{%
5344     \bbl@elt{english}{0}{\hyphen.tex}}%
5345     \bbl@elt{USenglish}{0}{}%
5346 \else
5347 \global\let\bbl@languages@format\bbl@languages
5348 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5349     \ifnum#2>\z@\else
5350     \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5351     \fi}%
5352 \xdef\bbl@languages{\bbl@languages}%
5353 \fi
5354 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}} % Define flags
5355 \bbl@languages
5356 \openin\bbl@readstream=language.dat
5357 \ifeof\bbl@readstream
5358     \bbl@warning{I couldn't find language.dat. No additional\\%
5359                 patterns loaded. Reported}%
5360 \else
5361     \loop
5362         \endlinechar\m@ne
5363         \read\bbl@readstream to \bbl@line
5364         \endlinechar\^^M
5365         \if T\ifeof\bbl@readstream F\fi T\relax
5366         \ifx\bbl@line\@empty\else
5367             \edef\bbl@line{\bbl@line\space\space\space}%
5368             \expandafter\bbl@process@line\bbl@line\relax
5369         \fi
5370     \repeat
5371 \fi
5372 \closein\bbl@readstream
5373 \endgroup
5374 \bbl@trace{Macros for reading patterns files}
5375 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
5376 \ifx\babelcatcodetablenum\undefined
5377 \ifx\newcatcodetable\undefined
5378     \def\babelcatcodetablenum{5211}
5379     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5380 \else
5381     \newcatcodetable\babelcatcodetablenum
5382     \newcatcodetable\bbl@pattcodes
5383 \fi
5384 \else
5385     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5386 \fi
5387 \def\bbl@luapatterns#1#2{%
5388     \bbl@get@enc#1: :@@@
5389     \setbox\z@\hbox\bgroup
5390     \begingroup
5391         \savecatcodetable\babelcatcodetablenum\relax
5392         \initcatcodetable\bbl@pattcodes\relax
5393         \catcodetable\bbl@pattcodes\relax
5394         \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5395         \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
5396         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12

```

```

5397     \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5398     \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5399     \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
5400     \input #1\relax
5401     \catcodetable\babelcatcodetablenum\relax
5402 \endgroup
5403 \def\bb@tempa{#2}%
5404 \ifx\bb@tempa\@empty\else
5405     \input #2\relax
5406 \fi
5407 \egroup}%
5408 \def\bb@patterns@lua#1{%
5409 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5410 \csname l@#1\endcsname
5411 \edef\bb@tempa{#1}%
5412 \else
5413 \csname l@#1:\f@encoding\endcsname
5414 \edef\bb@tempa{#1:\f@encoding}%
5415 \fi\relax
5416 \namedef\lu@texhyphen@loaded@the\language}{}% Temp
5417 \ifundefined{bb@hyphendata@the\language}%
5418 {\def\bb@elt##1##2##3##4{%
5419     \ifnum##2=\csname l@bb@tempa\endcsname % #2=spanish, dutch:OT1...
5420     \def\bb@tempb{##3}%
5421     \ifx\bb@tempb\@empty\else % if not a synonym
5422     \def\bb@tempc{{##3}{##4}}%
5423     \fi
5424     \bb@csarg\xdef{hyphendata@##2}{\bb@tempc}%
5425     \fi}%
5426 \bb@languages
5427 \ifundefined{bb@hyphendata@the\language}%
5428 {\bb@info{No hyphenation patterns were set for\%
5429     language '\bb@tempa'. Reported}}%
5430 {\expandafter\expandafter\expandafter\bb@luapatterns
5431     \csname bb@hyphendata@the\language\endcsname}}}%
5432 \endinput\fi

```

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

```

5433 \ifx\DisableBabelHook\@undefined
5434 \AddBabelHook{luatex}{everylanguage}{%
5435     \def\process@language##1##2##3{%
5436     \def\process@line####1####2 ####3 ####4 {}}
5437 \AddBabelHook{luatex}{loadpatterns}{%
5438     \input #1\relax
5439     \expandafter\gdef\csname bb@hyphendata@the\language\endcsname
5440     {#1}}}%
5441 \AddBabelHook{luatex}{loadexceptions}{%
5442     \input #1\relax
5443     \def\bb@tempb##1##2{{##1}{##1}}%
5444     \expandafter\xdef\csname bb@hyphendata@the\language\endcsname
5445     {\expandafter\expandafter\expandafter\bb@tempb
5446     \csname bb@hyphendata@the\language\endcsname}}
5447 \endinput\fi

```

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

```

5448 \begingroup
5449 \catcode`\%=12
5450 \catcode`\`=12
5451 \catcode`\\"=12
5452 \catcode`\:=12
5453 \directlua{
5454 Babel.locale_props = Babel.locale_props or {}
5455 function Babel.lua_error(e, a)

```

```

5456 tex.print([[noexpand\csname bbl@error\endcsname]] ..
5457 e .. '}{' .. (a or '') .. '}{}{')
5458 end
5459
5460 function Babel.bytes(line)
5461   return line:gsub(".",
5462     function (chr) return unicode.utf8.char(string.byte(chr)) end)
5463 end
5464
5465 function Babel.priority_in_callback(name,description)
5466   for i,v in ipairs(luatexbase.callback_descriptions(name)) do
5467     if v == description then return i end
5468   end
5469   return false
5470 end
5471
5472 function Babel.begin_process_input()
5473   if luatexbase and luatexbase.add_to_callback then
5474     luatexbase.add_to_callback('process_input_buffer',
5475       Babel.bytes,'Babel.bytes')
5476   else
5477     Babel.callback = callback.find('process_input_buffer')
5478     callback.register('process_input_buffer',Babel.bytes)
5479   end
5480 end
5481 function Babel.end_process_input ()
5482   if luatexbase and luatexbase.remove_from_callback then
5483     luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5484   else
5485     callback.register('process_input_buffer',Babel.callback)
5486   end
5487 end
5488
5489 function Babel.str_to_nodes(fn, matches, base)
5490   local n, head, last
5491   if fn == nil then return nil end
5492   for s in string.utfvalues(fn(matches)) do
5493     if base.id == 7 then
5494       base = base.replace
5495     end
5496     n = node.copy(base)
5497     n.char = s
5498     if not head then
5499       head = n
5500     else
5501       last.next = n
5502     end
5503     last = n
5504   end
5505   return head
5506 end
5507
5508 Babel.linebreaking = Babel.linebreaking or {}
5509 Babel.linebreaking.before = {}
5510 Babel.linebreaking.after = {}
5511 Babel.locale = {}
5512 function Babel.linebreaking.add_before(func, pos)
5513   tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5514   if pos == nil then
5515     table.insert(Babel.linebreaking.before, func)
5516   else
5517     table.insert(Babel.linebreaking.before, pos, func)
5518   end

```

```

5519 end
5520 function Babel.linebreaking.add_after(func)
5521   tex.print([[noexpand\csname bbl@lua hyphenate\endcsname]])
5522   table.insert(Babel.linebreaking.after, func)
5523 end
5524
5525 function Babel.addpatterns(pp, lg)
5526   local lg = lang.new(lg)
5527   local pats = lang.patterns(lg) or ''
5528   lang.clear_patterns(lg)
5529   for p in pp:gmatch('[^%s]+') do
5530     ss = ''
5531     for i in string.utfcharacters(p:gsub('%d', '')) do
5532       ss = ss .. '%d?' .. i
5533     end
5534     ss = ss:gsub('^%d%?%.', '%%.') .. '%d?'
5535     ss = ss:gsub('%.%d%?$', '%%.')
5536     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5537     if n == 0 then
5538       tex.sprint(
5539         [[\string\csname\space bbl@info\endcsname{New pattern: }
5540         .. p .. [{}]])
5541       pats = pats .. ' ' .. p
5542     else
5543       tex.sprint(
5544         [[\string\csname\space bbl@info\endcsname{Renew pattern: }
5545         .. p .. [{}]])
5546     end
5547   end
5548   lang.patterns(lg, pats)
5549 end
5550
5551 Babel.characters = Babel.characters or {}
5552 Babel.ranges = Babel.ranges or {}
5553 function Babel.hlist_has_bidi(head)
5554   local has_bidi = false
5555   local ranges = Babel.ranges
5556   for item in node.traverse(head) do
5557     if item.id == node.id'glyph' then
5558       local itemchar = item.char
5559       local chardata = Babel.characters[itemchar]
5560       local dir = chardata and chardata.d or nil
5561       if not dir then
5562         for nn, et in ipairs(ranges) do
5563           if itemchar < et[1] then
5564             break
5565           elseif itemchar <= et[2] then
5566             dir = et[3]
5567             break
5568           end
5569         end
5570       end
5571       if dir and (dir == 'al' or dir == 'r') then
5572         has_bidi = true
5573       end
5574     end
5575   end
5576   return has_bidi
5577 end
5578 function Babel.set_chrnges_b (script, chrng)
5579   if chrng == '' then return end
5580   texio.write('Replacing ' .. script .. ' script ranges')
5581   Babel.script_blocks[script] = {}

```

```

5582     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5583         table.insert(
5584             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5585     end
5586 end
5587
5588 function Babel.discard_sublr(str)
5589     if str:find( [[\string\indexentry]] ) and
5590         str:find( [[\string\babelsublr]] ) then
5591         str = str:gsub( [[\string\babelsublr*s*(%b{})]],
5592             function(m) return m:sub(2,-2) end )
5593     end
5594     return str
5595 end
5596 }
5597 \endgroup
5598 \ifx\newattribute\@undefined\else % Test for plain
5599     \newattribute\bbl@attr@locale % DL4
5600     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5601     \AddBabelHook{luatex}{beforeextras}{%
5602         \setattribute\bbl@attr@locale\localeid}
5603 \fi
5604 %
5605 \def\BabelStringsDefault{unicode}
5606 \let\luabbl@stop\relax
5607 \AddBabelHook{luatex}{encodedcommands}{%
5608     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5609     \ifx\bbl@tempa\bbl@tempb\else
5610         \directlua{Babel.begin_process_input()}%
5611         \def\luabbl@stop{%
5612             \directlua{Babel.end_process_input()}}%
5613     \fi}%
5614 \AddBabelHook{luatex}{stopcommands}{%
5615     \luabbl@stop
5616     \let\luabbl@stop\relax}
5617 %
5618 \AddBabelHook{luatex}{patterns}{%
5619     \@ifundefined{bbl@hyphendata@the\language}%
5620     {\def\bbl@elt##1##2##3##4{%
5621         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5622         \def\bbl@tempb{##3}%
5623         \ifx\bbl@tempb\@empty\else % if not a synonymous
5624             \def\bbl@tempc{##3}{##4}%
5625         \fi
5626         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5627     \fi}%
5628     \bbl@languages
5629     \@ifundefined{bbl@hyphendata@the\language}%
5630     {\bbl@info{No hyphenation patterns were set for\%
5631         language '#2'. Reported}}%
5632     {\expandafter\expandafter\expandafter\bbl@luapatterns
5633         \csname bbl@hyphendata@the\language\endcsname}}}%
5634 \@ifundefined{bbl@patterns@}{%
5635     \begingroup
5636         \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5637         \ifin@else
5638             \ifx\bbl@patterns@\@empty\else
5639                 \directlua{ Babel.addpatterns(
5640                     [[\bbl@patterns@]], \number\language) }%
5641             \fi
5642             \@ifundefined{bbl@patterns@#1}%
5643             \@empty
5644             {\directlua{ Babel.addpatterns(

```



```

5645         [[\space\csname bbl@patterns@#1\endcsname]],
5646         \number\language) } }%
5647     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5648     \fi
5649 \endgroup}%
5650 \bbl@exp{%
5651     \bbl@ifunset{\bbl@prehc@language}{ }%
5652     {\bbl@ifblank{\bbl@cs{prehc@language}}{ }%
5653     {\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.

```

5654 \@onlypreamble\babelpatterns
5655 \AtEndOfPackage{%
5656   \newcommand\babelpatterns[2][\@empty]{%
5657     \ifx\bbl@patterns@relax
5658       \let\bbl@patterns@empty
5659     \fi
5660     \ifx\bbl@pttnlist@empty\else
5661       \bbl@warning{%
5662         You must not intermingle \string\selectlanguage\space and\%
5663         \string\babelpatterns\space or some patterns will not\%
5664         be taken into account. Reported}%
5665       \fi
5666       \ifx@empty#1%
5667         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5668       \else
5669         \edef\bbl@tempb{\zap@space#1 \@empty}%
5670         \bbl@for\bbl@tempa\bbl@tempb{%
5671           \bbl@fixname\bbl@tempa
5672           \bbl@iflanguage\bbl@tempa{%
5673             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5674               \@ifundefined{\bbl@patterns@\bbl@tempa}{%
5675                 \@empty
5676                 {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5677               #2}}%
5678           \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.

```

5679 \def\bbl@intraspace#1 #2 #3\@{%
5680   \directlua{
5681     Babel.intraspaces = Babel.intraspaces or {}
5682     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5683     {b = #1, p = #2, m = #3}
5684     Babel.locale_props[\the\localeid].intraspace = %
5685     {b = #1, p = #2, m = #3}
5686   }}
5687 \def\bbl@intrapenalty#1\@{%
5688   \directlua{
5689     Babel.intrapenalties = Babel.intrapenalties or {}
5690     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5691     Babel.locale_props[\the\localeid].intrapenalty = #1
5692   }}
5693 \begingroup
5694 \catcode`\%=12
5695 \catcode`\&=14

```

```

5696 \catcode`\'=12
5697 \catcode`\-=12
5698 \gdef\bbl@seaintraspace{&
5699 \let\bbl@seaintraspace\relax
5700 \directlua{
5701   Babel.sea_enabled = true
5702   Babel.sea_ranges = Babel.sea_ranges or {}
5703   function Babel.set_chranges (script, chrng)
5704     local c = 0
5705     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.-)%s') do
5706       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5707       c = c + 1
5708     end
5709   end
5710   function Babel.sea_disc_to_space (head)
5711     local sea_ranges = Babel.sea_ranges
5712     local last_char = nil
5713     local quad = 655360      &% 10 pt = 655360 = 10 * 65536
5714     for item in node.traverse(head) do
5715       local i = item.id
5716       if i == node.id'glyph' then
5717         last_char = item
5718       elseif i == 7 and item.subtype == 3 and last_char
5719         and last_char.char > 0x0C99 then
5720         quad = font.getfont(last_char.font).size
5721         for lg, rg in pairs(sea_ranges) do
5722           if last_char.char > rg[1] and last_char.char < rg[2] then
5723             lg = lg:sub(1, 4) &% Remove trailing number of, e.g., Cyril
5724             local intraspace = Babel.intraspaces[lg]
5725             local intrapenalty = Babel.intrapenalties[lg]
5726             local n
5727             if intrapenalty ~= 0 then
5728               n = node.new(14, 0)      &% penalty
5729               n.penalty = intrapenalty
5730               node.insert_before(head, item, n)
5731             end
5732             n = node.new(12, 13)      &% (glue, spaceskip)
5733             node.setglue(n, intraspace.b * quad,
5734               intraspace.p * quad,
5735               intraspace.m * quad)
5736             node.insert_before(head, item, n)
5737             node.remove(head, item)
5738           end
5739         end
5740       end
5741     end
5742   end
5743 }&
5744 \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.

```

5745 \catcode`\%=14
5746 \gdef\bbl@cjkkintraspace{%
5747 \let\bbl@cjkkintraspace\relax
5748 \directlua{
5749   require('babel-data-cjk.lua')

```

```

5750 Babel.cjk_enabled = true
5751 function Babel.cjk_linebreak(head)
5752     local GLYPH = node.id'glyph'
5753     local last_char = nil
5754     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5755     local last_class = nil
5756     local last_lang = nil
5757     for item in node.traverse(head) do
5758         if item.id == GLYPH then
5759             local lang = item.lang
5760             local LOCALE = node.get_attribute(item,
5761                 Babel.attr_locale)
5762             local props = Babel.locale_props[LOCALE] or {}
5763             local class = Babel.cjk_class[item.char].c
5764             if props.cjk_quotes and props.cjk_quotes[item.char] then
5765                 class = props.cjk_quotes[item.char]
5766             end
5767             if class == 'cp' then class = 'cl' % ) as CL
5768             elseif class == 'id' then class = 'I'
5769             elseif class == 'cj' then class = 'I' % loose
5770             end
5771             local br = 0
5772             if class and last_class and Babel.cjk_breaks[last_class][class] then
5773                 br = Babel.cjk_breaks[last_class][class]
5774             end
5775             if br == 1 and props.linebreak == 'c' and
5776                 lang ~= \the\l@nohyphenation\space and
5777                 last_lang ~= \the\l@nohyphenation then
5778                 local intrapenalty = props.intrapenalty
5779                 if intrapenalty ~= 0 then
5780                     local n = node.new(14, 0)      % penalty
5781                     n.penalty = intrapenalty
5782                     node.insert_before(head, item, n)
5783                 end
5784                 local intraspace = props.intraspace
5785                 local n = node.new(12, 13)      % (glue, spaceskip)
5786                 node.setglue(n, intraspace.b * quad,
5787                     intraspace.p * quad,
5788                     intraspace.m * quad)
5789                 node.insert_before(head, item, n)
5790             end
5791             if font.getfont(item.font) then
5792                 quad = font.getfont(item.font).size
5793             end
5794             last_class = class
5795             last_lang = lang
5796             else % if penalty, glue or anything else
5797                 last_class = nil
5798             end
5799         end
5800         lang.hyphenate(head)
5801     end
5802 }%
5803 \bbl@luahyphenate}
5804 \gdef\bbl@luahyphenate{%
5805 \let\bbl@luahyphenate\relax
5806 \directlua{
5807     luatexbase.add_to_callback('hyphenate',
5808     function (head, tail)
5809         if Babel.linebreaking.before then
5810             for k, func in ipairs(Babel.linebreaking.before) do
5811                 func(head)
5812             end

```

```

5813     end
5814     lang.hyphenate(head)
5815     if Babel.cjk_enabled then
5816         Babel.cjk_linebreak(head)
5817     end
5818     if Babel.linebreaking.after then
5819         for k, func in ipairs(Babel.linebreaking.after) do
5820             func(head)
5821         end
5822     end
5823     if Babel.set_hboxed then
5824         Babel.set_hboxed(head)
5825     end
5826     if Babel.sea_enabled then
5827         Babel.sea_disc_to_space(head)
5828     end
5829 end,
5830 'Babel.hyphenate')
5831 }}
5832 \endgroup
5833 %
5834 \def\bbl@provide@intraspace{%
5835   \bbl@ifunset{\bbl@intsp@\languagename}{}%
5836   {\expandafter\ifx\cename\bbl@intsp@\languagename\endcename\@empty\else
5837     \bbl@xin@{/c}{/\bbl@c{l}{lnbrk}}%
5838     \ifin@           % cjk
5839     \bbl@cjkintraspace
5840     \directlua{
5841       Babel.locale_props = Babel.locale_props or {}
5842       Babel.locale_props[\the\localeid].linebreak = 'c'
5843     }%
5844     \bbl@exp{\bbl@intraspace\bbl@c{l}{intsp}}\@%
5845     \ifx\bbl@KVP@intrapenalty\@nnil
5846       \bbl@intrapenalty0\@@
5847     \fi
5848   \else           % sea
5849     \bbl@seaintraspace
5850     \bbl@exp{\bbl@intraspace\bbl@c{l}{intsp}}\@%
5851     \directlua{
5852       Babel.sea_ranges = Babel.sea_ranges or {}
5853       Babel.set_chranges('\bbl@c{l}{sbcp}',
5854         '\bbl@c{l}{chrng}')
5855     }%
5856     \ifx\bbl@KVP@intrapenalty\@nnil
5857       \bbl@intrapenalty0\@@
5858     \fi
5859   \fi
5860 \fi
5861 \ifx\bbl@KVP@intrapenalty\@nnil\else
5862   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5863 \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`.

```

5864 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5865 \def\bblar@chars{%
5866   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5867   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5868   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5869 \def\bblar@elongated{%
5870   0626,0628,062A,062B,0633,0634,0635,0636,063B,%

```

```

5871 063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5872 0649,064A}
5873 \begingroup
5874 \catcode`_ =11 \catcode`:=11
5875 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5876 \endgroup
5877 \gdef\bbl@arabicjust{%
5878 \let\bbl@arabicjust\relax
5879 \newattribute\bblar@kashida
5880 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5881 \bblar@kashida=\z@
5882 \bbl@patchfont{\bbl@parsejalt}}%
5883 \directlua{
5884 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5885 Babel.arabic.elong_map[\the\localeid] = {}
5886 luatexbase.add_to_callback('post_linebreak_filter',
5887 Babel.arabic.justify, 'Babel.arabic.justify')
5888 luatexbase.add_to_callback('hpack_filter',
5889 Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5890 }%

```

Save both node lists to make replacement.

```

5891 \def\bblar@fetchjalt#1#2#3#4{%
5892 \bbl@exp{\bbl@foreach#1}}{%
5893 \bbl@ifunset\bblar@JE@##1}%
5894 {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5895 {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5896 \directlua{%
5897 local last = nil
5898 for item in node.traverse(tex.box[0].head) do
5899 if item.id == node.id'glyph' and item.char > 0x600 and
5900 not (item.char == 0x200D) then
5901 last = item
5902 end
5903 end
5904 Babel.arabic.#3['##1#4'] = last.char
5905 }}

```

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

```

5906 \gdef\bbl@parsejalt{%
5907 \ifx\addfontfeature\undefined\else
5908 \bbl@xin@{/e}{/\bbl@cl{lbrk}}}%
5909 \ifin@
5910 \directlua{%
5911 if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5912 Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5913 tex.print([[string\csname\space bbl@parsejalti\endcsname]])
5914 end
5915 }%
5916 \fi
5917 \fi}
5918 \gdef\bbl@parsejalti{%
5919 \begingroup
5920 \let\bbl@parsejalt\relax % To avoid infinite loop
5921 \edef\bbl@tempb{\fontid\font}%
5922 \bblar@nofswarn
5923 \@nameuse{tracinglostchars}\z@
5924 \bblar@fetchjalt\bblar@elongated{}{from}}%
5925 \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5926 \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5927 \addfontfeature{RawFeature+=jalt}%
5928 % \namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5929 \bblar@fetchjalt\bblar@elongated{}{dest}}%

```

```

5930 \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5931 \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5932 \directlua{%
5933     for k, v in pairs(Babel.arabic.from) do
5934         if Babel.arabic.dest[k] and
5935             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5936             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5937                 [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5938         end
5939     end
5940 }%
5941 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5942 \begingroup
5943 \catcode`#=11
5944 \catcode`~=11
5945 \directlua{
5946
5947 Babel.arabic = Babel.arabic or {}
5948 Babel.arabic.from = {}
5949 Babel.arabic.dest = {}
5950 Babel.arabic.justify_factor = 0.95
5951 Babel.arabic.justify_enabled = true
5952 Babel.arabic.kashida_limit = -1
5953
5954 function Babel.arabic.justify(head)
5955     if not Babel.arabic.justify_enabled then return head end
5956     for line in node.traverse_id(node.id'hlist', head) do
5957         Babel.arabic.justify_hlist(head, line)
5958     end
5959     % In case the very first item is a line (eg, in \vbox):
5960     while head.prev do head = head.prev end
5961     return head
5962 end
5963
5964 function Babel.arabic.justify_hbox(head, gc, size, pack)
5965     local has_inf = false
5966     if Babel.arabic.justify_enabled and pack == 'exactly' then
5967         for n in node.traverse_id(12, head) do
5968             if n.stretch_order > 0 then has_inf = true end
5969         end
5970         if not has_inf then
5971             Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5972         end
5973     end
5974     return head
5975 end
5976
5977 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5978     local d, new
5979     local k_list, k_item, pos_inline
5980     local width, width_new, full, k_curr, wt_pos, goal, shift
5981     local subst_done = false
5982     local elong_map = Babel.arabic.elong_map
5983     local cnt
5984     local last_line
5985     local GLYPH = node.id'glyph'
5986     local KASHIDA = Babel.attr_kashida
5987     local LOCALE = Babel.attr_locale
5988
5989     if line == nil then
5990         line = {}

```

```

5991     line.glue_sign = 1
5992     line.glue_order = 0
5993     line.head = head
5994     line.shift = 0
5995     line.width = size
5996 end
5997
5998 % Exclude last line.
5999 if ((line.next ~= nil or line.glue_sign == 1) and line.glue_order == 0) then
6000     elongs = {}      % Stores elongated candidates of each line
6001     k_list = {}      % And all letters with kashida
6002     pos_inline = 0  % Not yet used
6003
6004     for n in node.traverse_id(GLYPH, line.head) do
6005         pos_inline = pos_inline + 1 % To find where it is. Not used.
6006
6007         % Elongated glyphs
6008         if elong_map then
6009             local locale = node.get_attribute(n, LOCALE)
6010             if elong_map[locale] and elong_map[locale][n.font] and
6011                 elong_map[locale][n.font][n.char] then
6012                 table.insert(elongs, {node = n, locale = locale} )
6013                 node.set_attribute(n.prev, KASHIDA, 0)
6014             end
6015         end
6016
6017         % Tatwil. First create a list of nodes marked with kashida. The
6018         % rest of nodes can be ignored. The list of used weights is build
6019         % when transforms with the key kashida= are declared.
6020         if Babel.kashida_wts then
6021             local k_wt = node.get_attribute(n, KASHIDA)
6022             if k_wt > 0 then % todo. parameter for multi inserts
6023                 table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
6024             end
6025         end
6026
6027     end % of node.traverse_id
6028
6029     if #elongs == 0 and #k_list == 0 then goto next_line end
6030     full = line.width
6031     shift = line.shift
6032     goal = full * Babel.arabic.justify_factor % A bit crude
6033     width = node.dimensions(line.head)      % The 'natural' width
6034
6035     % == Elongated ==
6036     % Original idea taken from 'chickenize'
6037     while (#elongs > 0 and width < goal) do
6038         subst_done = true
6039         local x = #elongs
6040         local curr = elongs[x].node
6041         local oldchar = curr.char
6042         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
6043         width = node.dimensions(line.head) % Check if the line is too wide
6044         % Substitute back if the line would be too wide and break:
6045         if width > goal then
6046             curr.char = oldchar
6047             break
6048         end
6049         % If continue, pop the just substituted node from the list:
6050         table.remove(elongs, x)
6051     end
6052
6053     % == Tatwil ==

```

```

6054 % Traverse the kashida node list so many times as required, until
6055 % the line is filled. The first pass adds a tatweel after each
6056 % node with kashida in the line, the second pass adds another one,
6057 % and so on. In each pass, add first the kashida with the highest
6058 % weight, then with lower weight and so on.
6059 if #k_list == 0 then goto next_line end
6060
6061 width = node.dimensions(line.head) % The 'natural' width
6062 k_curr = #k_list % Traverse backwards, from the end
6063 wt_pos = 1
6064
6065 while width < goal do
6066     subst_done = true
6067     k_item = k_list[k_curr].node
6068     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
6069         d = node.copy(k_item)
6070         d.char = 0x0640
6071         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
6072         d.xoffset = 0
6073         line.head, new = node.insert_after(line.head, k_item, d)
6074         width_new = node.dimensions(line.head)
6075         if width > goal or width == width_new then
6076             node.remove(line.head, new) % Better compute before
6077             break
6078         end
6079         if Babel.fix_diacr then
6080             Babel.fix_diacr(k_item.next)
6081         end
6082         width = width_new
6083     end
6084     if k_curr == 1 then
6085         k_curr = #k_list
6086         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
6087     else
6088         k_curr = k_curr - 1
6089     end
6090 end
6091
6092 % Limit the number of tatweel by removing them. Not very efficient,
6093 % but it does the job in a quite predictable way.
6094 if Babel.arabic.kashida_limit > -1 then
6095     cnt = 0
6096     for n in node.traverse_id(GLYPH, line.head) do
6097         if n.char == 0x0640 then
6098             cnt = cnt + 1
6099             if cnt > Babel.arabic.kashida_limit then
6100                 node.remove(line.head, n)
6101             end
6102         else
6103             cnt = 0
6104         end
6105     end
6106 end
6107
6108 ::next_line::
6109
6110 % Must take into account marks and ins, see luatex manual.
6111 % Have to be executed only if there are changes. Investigate
6112 % what's going on exactly.
6113 if subst_done and not gc then
6114     d = node.hpack(line.head, full, 'exactly')
6115     d.shift = shift
6116     node.insert_before(head, line, d)

```



```

6117     node.remove(head, line)
6118   end
6119 end % if process line
6120 end
6121 }
6122 \endgroup
6123 \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`.

```

6124 \def\bbl@scr@node@list{%
6125   ,Armenian,Coptic,Cyrillic,Georgian,,Glagolitic,Gothic,%
6126   ,Greek,Latin,Old Church Slavonic Cyrillic,}
6127 \ifnum\bbl@bidimode=102 % bidi-r
6128   \bbl@add\bbl@scr@node@list{Arabic,Hebrew,Syriac}
6129 \fi
6130 \def\bbl@set@renderer{%
6131   \bbl@xin@{\bbl@cl{sname}}{\bbl@scr@node@list}%
6132   \ifin@
6133     \let\bbl@unset@renderer\relax
6134   \else
6135     \bbl@exp{%
6136       \def\\bbl@unset@renderer{%
6137         \def<g__fontspec_default_fontopts_clist>{%
6138           \[g__fontspec_default_fontopts_clist]}%
6139         \def<g__fontspec_default_fontopts_clist>{%
6140           Renderer=Harfbuzz,\[g__fontspec_default_fontopts_clist]}%
6141       \fi}
6142 <@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.

There are two situations where the replacement is not carried out: either the `letters` option has been set and the character is not a letter (in the \TeX sense), or the current script is the same as the new one.

```

6143 \directlua{% DL6
6144 Babel.script_blocks = {
6145   ['dflt'] = {},
6146   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6147             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6148   ['Armn'] = {{0x0530, 0x058F}},
6149   ['Beng'] = {{0x0980, 0x09FF}},
6150   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6151   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6152   ['Cyr'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6153             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6154   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6155   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6156             {0xAB00, 0xAB2F}},

```

```

6157 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6158 % Don't follow strictly Unicode, which places some Coptic letters in
6159 % the 'Greek and Coptic' block
6160 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6161 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6162             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6163             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6164             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6165             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6166             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6167 ['Hebr'] = {{0x0590, 0x05FF},
6168             {0xFB1F, 0xFB4E}}, % <- Includes some <reserved>
6169 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6170             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6171 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6172 ['Knda'] = {{0x0C80, 0x0CFF}},
6173 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6174             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6175             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6176 ['Lao'] = {{0x0E80, 0x0EFF}},
6177 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6178             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6179             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6180 ['Mahj'] = {{0x11150, 0x1117F}},
6181 ['Mlym'] = {{0x0D00, 0x0D7F}},
6182 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6183 ['Orya'] = {{0x0B00, 0x0B7F}},
6184 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
6185 ['Sycr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6186 ['Taml'] = {{0x0B80, 0x0BFF}},
6187 ['Telu'] = {{0x0C00, 0x0C7F}},
6188 ['Tfng'] = {{0x2D30, 0x2D7F}},
6189 ['Thai'] = {{0x0E00, 0x0E7F}},
6190 ['Tibt'] = {{0x0F00, 0x0FFF}},
6191 ['Vaii'] = {{0xA500, 0xA63F}},
6192 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6193 }
6194
6195 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
6196 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6197 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6198
6199 function Babel.locale_map(head)
6200   if not Babel.locale_mapped then return head end
6201
6202   local LOCALE = Babel.attr_locale
6203   local GLYPH = node.id('glyph')
6204   local inmath = false
6205   local toloc_save
6206   for item in node.traverse(head) do
6207     local toloc
6208     if not inmath and item.id == GLYPH then
6209       % Optimization: build a table with the chars found
6210       if Babel.chr_to_loc[item.char] then
6211         toloc = Babel.chr_to_loc[item.char]
6212       else
6213         for lc, maps in pairs(Babel.loc_to_scr) do
6214           for _, rg in pairs(maps) do
6215             if item.char >= rg[1] and item.char <= rg[2] then
6216               Babel.chr_to_loc[item.char] = lc
6217               toloc = lc
6218               break
6219             end
6220           end
6221         end
6222       end
6223     end
6224     toloc_save = toloc
6225     item[LOCALE] = toloc
6226     if item.id == GLYPH then inmath = true end
6227   end
6228   Babel.locale_mapped = true
6229 end

```

```

6220         end
6221     end
6222     % Treat composite chars in a different fashion, because they
6223     % 'inherit' the previous locale.
6224     if (item.char >= 0x0300 and item.char <= 0x036F) or
6225        (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6226        (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6227         Babel.chr_to_loc[item.char] = -2000
6228         toloc = -2000
6229     end
6230     if not toloc then
6231         Babel.chr_to_loc[item.char] = -1000
6232     end
6233     end
6234     if toloc == -2000 then
6235         toloc = toloc_save
6236     elseif toloc == -1000 then
6237         toloc = nil
6238     end
6239     if toloc and Babel.locale_props[toloc] and
6240        Babel.locale_props[toloc].letters and
6241        tex.getcatcode(item.char) \string-= 11 then
6242         toloc = nil
6243     end
6244     if toloc and Babel.locale_props[toloc].script
6245        and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6246        and Babel.locale_props[toloc].script ==
6247        Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6248         toloc = nil
6249     end
6250     if toloc then
6251         if Babel.locale_props[toloc].lg then
6252             item.lang = Babel.locale_props[toloc].lg
6253             node.set_attribute(item, LOCALE, toloc)
6254         end
6255         if Babel.locale_props[toloc]['/'..item.font] then
6256             item.font = Babel.locale_props[toloc]['/'..item.font]
6257         end
6258     end
6259     toloc_save = toloc
6260 elseif not inmath and item.id == 7 then % Apply recursively
6261     item.replace = item.replace and Babel.locale_map(item.replace)
6262     item.pre     = item.pre and Babel.locale_map(item.pre)
6263     item.post    = item.post and Babel.locale_map(item.post)
6264 elseif item.id == node.id'math' then
6265     inmath = (item.subtype == 0)
6266 end
6267 end
6268 return head
6269 end
6270 }

```

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

```

6271 \newcommand\babelcharproperty[1]{%
6272   \count@=#1\relax
6273   \ifvmode
6274     \expandafter\bbl@chprop
6275   \else
6276     \bbl@error{charproperty-only-vertical}{#1}{#1}%
6277   \fi}
6278 \newcommand\bbl@chprop[3][\the\count@]{%
6279   \@tempcnta=#1\relax

```

```

6280 \bbl@ifunset{bbl@chprop@#2}% {unknown-char-property}
6281   {\bbl@error{unknown-char-property}{#2}}}%
6282   }%
6283 \loop
6284   \bbl@cs{chprop@#2}{#3}%
6285   \ifnum\count@<\@tempcnta
6286     \advance\count@\@ne
6287   \repeat}
6288 %
6289 \def\bbl@chprop@direction#1{%
6290   \directlua{
6291     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6292     Babel.characters[\the\count@]['d'] = '#1'
6293   }}
6294 \let\bbl@chprop@bc\bbl@chprop@direction
6295 %
6296 \def\bbl@chprop@mirror#1{%
6297   \directlua{
6298     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6299     Babel.characters[\the\count@]['m'] = '\number#1'
6300   }}
6301 \let\bbl@chprop@bmg\bbl@chprop@mirror
6302 %
6303 \def\bbl@chprop@linebreak#1{%
6304   \directlua{
6305     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6306     Babel.cjk_characters[\the\count@]['c'] = '#1'
6307   }}
6308 \let\bbl@chprop@lb\bbl@chprop@linebreak
6309 %
6310 \def\bbl@chprop@locale#1{%
6311   \directlua{
6312     Babel.chr_to_loc = Babel.chr_to_loc or {}
6313     Babel.chr_to_loc[\the\count@] =
6314       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
6315   }}

```

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.

```

6316 \directlua{% DL7
6317   Babel.nohyphenation = \the\@nohyphenation
6318 }

```

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).

```

6319 \begingroup
6320 \catcode`\~ =12
6321 \catcode`\% =12
6322 \catcode`\& =14
6323 \catcode`\| =12
6324 \gdef\babelprehyphenation{%&%
6325   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]]}
6326 \gdef\babelposthyphenation{%&%
6327   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]]}
6328 %
6329 \gdef\bbl@settransform#1[#2]#3#4#5{%&%
6330   \ifcase#1

```

```

6331 \bbl@activateprehyphen
6332 \or
6333 \bbl@activateposthyphen
6334 \fi
6335 \begingroup
6336 \def\babeltempa{\bbl@add@list\babeltempb}&%
6337 \let\babeltempb\empty
6338 \def\bbl@tempa{#5}&%
6339 \bbl@replace\bbl@tempa{,}{,}&% TODO. Ugly trick to preserve {}
6340 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6341 \bbl@ifsamestring{##1}{remove}&%
6342 {\bbl@add@list\babeltempb{nil}}&%
6343 {\directlua{
6344 local rep = [=##1]=]
6345 local three_args = '%s*=%s*([%-d%.%a{|}]+)%s+([%-d%.%a{|}]+)%s+([%-d%.%a{|}]+)'
6346 &% Numeric passes directly: kern, penalty...
6347 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6348 rep = rep:gsub('^%s*(insert)%s*', ', 'insert = true, ')
6349 rep = rep:gsub('^%s*(after)%s*', ', 'after = true, ')
6350 rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6351 rep = rep:gsub('node%s*=%s*(%a)%s*(%a*)', Babel.capture_node)
6352 rep = rep:gsub(' (norule)' .. three_args,
6353 'norule = {' .. '%2, %3, %4' .. '}')
6354 if #1 == 0 or #1 == 2 then
6355 rep = rep:gsub(' (space)' .. three_args,
6356 'space = {' .. '%2, %3, %4' .. '}')
6357 rep = rep:gsub(' (spacefactor)' .. three_args,
6358 'spacefactor = {' .. '%2, %3, %4' .. '}')
6359 rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6360 &% Transform values
6361 rep, n = rep:gsub( '{{[%a%-%.]+}|([%a%_%.]+)}',
6362 function(v,d)
6363 return string.format (
6364 '\the\csname bbl@id@@#3\endcsname,"%s",%s}',
6365 v,
6366 load( 'return Babel.locale_props'..
6367 '\the\csname bbl@id@@#3\endcsname.' .. d)() )
6368 end )
6369 rep, n = rep:gsub( '{{[%a%-%.]+}|([%-d%.]+)}',
6370 '\the\csname bbl@id@@#3\endcsname,"%1",%2}')
6371 end
6372 if #1 == 1 then
6373 rep = rep:gsub( '(no)%s*=%s*([^\s,]*)', Babel.capture_func)
6374 rep = rep:gsub( '(pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6375 rep = rep:gsub( '(post)%s*=%s*([^\s,]*)', Babel.capture_func)
6376 end
6377 tex.print([\string\babeltempa{[]} .. rep .. [[]]])
6378 }}&%
6379 \bbl@foreach\babeltempb{&%
6380 \bbl@forkv{##1}}&%
6381 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,no,&%
6382 post,penalty,kashida,space,spacefactor,kern,node,after,norule,}&%
6383 \ifin@else
6384 \bbl@error{bad-transform-option}{###1}{}&%
6385 \fi}&%
6386 \let\bbl@kv@attribute\relax
6387 \let\bbl@kv@label\relax
6388 \let\bbl@kv@fonts\empty
6389 \let\bbl@kv@prepend\relax
6390 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6391 \ifx\bbl@kv@fonts\empty\else\bbl@settransformfont\fi
6392 \ifx\bbl@kv@attribute\relax
6393 \ifx\bbl@kv@label\relax\else

```

```

6394 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6395 \bbl@replace\bbl@kv@fonts{ }{,}&%
6396 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6397 \count@ \z@
6398 \def\bbl@elt##1##2##3{&%
6399 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6400 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6401 {\count@\@ne}&%
6402 {\bbl@error{font-conflict-transforms}{}}{}}}&%
6403 {}}&%
6404 \bbl@transfont@list
6405 \ifnum\count@=\z@
6406 \bbl@exp{\global\bbl@add\bbl@transfont@list
6407 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6408 \fi
6409 \bbl@ifunset{\bbl@kv@attribute}&%
6410 {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6411 {}&%
6412 \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6413 \fi
6414 \else
6415 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6416 \fi
6417 \directlua{
6418 local lbkr = Babel.linebreaking.replacements[#1]
6419 local u = unicode.utf8
6420 local id, attr, label
6421 if #1 == 0 then
6422 id = \the\csname bbl@id@#3\endcsname\space
6423 else
6424 id = \the\csname l@#3\endcsname\space
6425 end
6426 \ifx\bbl@kv@attribute\relax
6427 attr = -1
6428 \else
6429 attr = luatexbase.registernumber'\bbl@kv@attribute'
6430 \fi
6431 \ifx\bbl@kv@label\relax\else &% Same refs:
6432 label = [==[\bbl@kv@label]==]
6433 \fi
6434 &% Convert pattern:
6435 local patt = string.gsub([==[#4]==], '%s', '')
6436 if #1 == 0 then
6437 patt = string.gsub(patt, '|', ' ')
6438 end
6439 if not u.find(patt, '()', nil, true) then
6440 patt = '()' .. patt .. '()'
6441 end
6442 patt = string.gsub(patt, '%(%)^', '^()')
6443 patt = string.gsub(patt, '%$(%)', '()$')
6444 patt = u.gsub(patt, '{(.)}',
6445 function (n)
6446 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6447 end)
6448 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6449 function (n)
6450 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6451 end)
6452 lbkr[id] = lbkr[id] or {}
6453 table.insert(lbkr[id], \ifx\bbl@kv@prepend\relax\else 1,\fi
6454 { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6455 }&%
6456 \endgroup}

```

```

6457 \endgroup
6458 %
6459 \let\bbl@transfont@list\@empty
6460 \def\bbl@settransfont{%
6461   \global\let\bbl@settransfont\relax % Execute only once
6462   \gdef\bbl@transfont{%
6463     \def\bbl@elt####1####2####3{%
6464       \bbl@ifblank{####3}%
6465       {\count@tw@}% Do nothing if no fonts
6466       {\count@z@
6467         \bbl@vforeach{####3}{%
6468           \def\bbl@tempd{#####1}%
6469           \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6470           \ifx\bbl@tempd\bbl@tempe
6471             \count@\@ne
6472           \else\ifx\bbl@tempd\bbl@transfam
6473             \count@\@ne
6474           \fi\fi}%
6475       \ifcase\count@
6476         \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
6477       \or
6478         \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
6479       \fi}}%
6480     \bbl@transfont@list}%
6481 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6482 \gdef\bbl@transfam{-unknown-}%
6483 \bbl@foreach\bbl@font@fams{%
6484   \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6485   \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6486   {\xdef\bbl@transfam{##1}}%
6487   {}}
6488 %
6489 \DeclareRobustCommand\enablelocaletransform[1]{%
6490   \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6491   {\bbl@error{transform-not-available}{#1}{}}%
6492   {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6493 \DeclareRobustCommand\disablelocaletransform[1]{%
6494   \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6495   {\bbl@error{transform-not-available-b}{#1}{}}%
6496   {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}

```

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

```

6497 \def\bbl@activateposthyphen{%
6498   \let\bbl@activateposthyphen\relax
6499   \ifx\bbl@attr@hboxed\undefined
6500     \newattribute\bbl@attr@hboxed
6501   \fi
6502   \directlua{
6503     require('babel-transforms.lua')
6504     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6505   }}
6506 \def\bbl@activateprehyphen{%
6507   \let\bbl@activateprehyphen\relax
6508   \ifx\bbl@attr@hboxed\undefined
6509     \newattribute\bbl@attr@hboxed
6510   \fi
6511   \directlua{
6512     require('babel-transforms.lua')
6513     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6514   }}
6515 \newcommand\SetTransformValue[3]{%
6516   \directlua{

```

```

6517   Babel.locale_props[\the\csname bbl@id@@#1\endcsname].vars["#2"] = #3
6518 }}

```

The code in `babel-t-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.

```

6519 \newcommand\ShowBabelTransforms[1]{%
6520   \bbl@activateprehyphen
6521   \bbl@activateposthyphen
6522   \begingroup
6523   \directlua{ Babel.show_transforms = true }%
6524   \setbox\z@\vbox{#1}%
6525   \directlua{ Babel.show_transforms = false }%
6526   \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.

```

6527 \newcommand\localeprehyphenation[1]{%
6528   \directlua{ Babel.string_prehyphenation([=#1]=], \the\localeid) }}

```

10.11.Bidi

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

```

6529 \def\bbl@activate@preotf{%
6530   \let\bbl@activate@preotf\relax % only once
6531   \directlua{
6532     function Babel.pre_otfload_v(head)
6533       if Babel.numbers and Babel.digits_mapped then
6534         head = Babel.numbers(head)
6535       end
6536       if Babel.bidi_enabled then
6537         head = Babel.bidi(head, false, dir)
6538       end
6539       return head
6540     end
6541     %
6542     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6543       if Babel.numbers and Babel.digits_mapped then
6544         head = Babel.numbers(head)
6545       end
6546       if Babel.bidi_enabled then
6547         head = Babel.bidi(head, false, dir)
6548       end
6549       return head
6550     end
6551     %
6552     luatexbase.add_to_callback('pre_linebreak_filter',
6553       Babel.pre_otfload_v,
6554       'Babel.pre_otfload_v',
6555       Babel.priority_in_callback('pre_linebreak_filter',
6556         'luaotfload.node_processor') or nil)
6557     %
6558     luatexbase.add_to_callback('hpack_filter',
6559       Babel.pre_otfload_h,
6560       'Babel.pre_otfload_h',
6561       Babel.priority_in_callback('hpack_filter',
6562         'luaotfload.node_processor') or nil)
6563   }}

```


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`.

```

6564 \ifnum\bbl@bidimode>\@ne % Any bidi= except default (=1)
6565 \let\bbl@beforeforeign\leavevmode
6566 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6567 \RequirePackage{luatexbase}
6568 \bbl@activate@preotf
6569 \directlua{
6570   require('babel-data-bidi.lua')
6571   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6572     require('babel-bidi-basic.lua')
6573   \or
6574     require('babel-bidi-basic-r.lua')
6575     table.insert(Babel.ranges, {0xE000, 0xF8FF, 'on'})
6576     table.insert(Babel.ranges, {0xF000, 0xFFFFD, 'on'})
6577     table.insert(Babel.ranges, {0x10000, 0x10FFFD, 'on'})
6578   \fi}
6579 \newattribute\bbl@attr@dir
6580 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6581 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6582 \fi
6583 %
6584 \chardef\bbl@thetextdir\z@
6585 \chardef\bbl@thepardir\z@
6586 \def\bbl@setluadir#1#2{% 1=\text/pardirection 2= 0:l 1:r 2:a}
6587 \ifcase#2\relax
6588   \ifcase#1\else#1=\z@\fi
6589 \else
6590   \ifcase#1#1=\@ne\fi
6591 \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`). These macro names are shared by the 3 engines, with different definitions.

```

6592 \def\bbl@thedir{0}
6593 \def\bbl@textdir#1{%
6594   \bbl@setluadir\textdirection{#1}%
6595   \chardef\bbl@thetextdir#1\relax
6596   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6597   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6598 \def\bbl@pardir#1{% Used twice
6599   \bbl@setluadir\pardirection{#1}%
6600   \chardef\bbl@thepardir#1\relax}
6601 \def\bbl@bodydir{\bbl@setluadir\bodydirection}% Used once
6602 \def\bbl@dirparastext{\pardirection=\textdirection\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.

```

6603 \ifnum\bbl@bidimode>\z@ % Any bidi=
6604 \def\bbl@insidemath{0}%
6605 \def\bbl@everymath{\def\bbl@insidemath{1}}
6606 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6607 \frozen@everymath\expandafter{%
6608   \expandafter\bbl@everymath\the\frozen@everymath}
6609 \frozen@everydisplay\expandafter{%
6610   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6611 \AtBeginDocument{
6612   \directlua{
6613     function Babel.math_box_dir(head)
6614       if not (token.get_macro('bbl@insidemath') == '0') then
6615         if Babel.hlist_has_bidi(head) then
6616           local d = node.new(node.id'dir')

```

```

6617         d.dir = '+TRT'
6618     for item in node.traverse(head) do
6619         if item.id == 11 or item.id == node.id'glyph' then
6620             head = node.insert_before(head, item, d)
6621             break
6622         end
6623     end
6624     local inmath = false
6625     for item in node.traverse(head) do
6626         if item.id == 11 then
6627             inmath = (item.subtype == 0)
6628         elseif not inmath then
6629             node.set_attribute(item,
6630                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6631         end
6632     end
6633 end
6634 end
6635 return head
6636 end
6637 luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6638     "Babel.math_box_dir", 0)
6639 if Babel.unset_atdir then
6640     luatexbase.add_to_callback("pre_linebreak_filter", Babel.unset_atdir,
6641         "Babel.unset_atdir")
6642     luatexbase.add_to_callback("hpack_filter", Babel.unset_atdir,
6643         "Babel.unset_atdir")
6644 end
6645 }}%
6646 \fi

```

Experimental. Tentative name.

```

6647 \DeclareRobustCommand\localebox[1]{%
6648     {\def\bbl@insidemath{0}%
6649         \mbox{\foreignlanguage{\language}\#1}}}

```

10.12 Layout

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

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

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

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

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

```

6650 \bbl@trace{Redefinitions for bidi layout}
6651 %
6652 <<(*More package options)>> ≡
6653 \chardef\bbl@eqnpos\z@
6654 \DeclareOption{leqno}{\chardef\bbl@eqnpos@ne}
6655 \DeclareOption{fleqn}{\chardef\bbl@eqnpos@tw@}

```

```

6656 <</More package options>>
6657 %
6658 \ifnum\bbbl@bidimode>\z@ % Any bidi=
6659 \matheqdirmode\@ne % Several luatex primitives.
6660 \mathemptydisplaymode\@ne % For fixes.
6661 \breakafterdirmode\@ne %
6662 \fixupboxesmode\@ne %
6663 \let\bbbl@eqnodir\relax
6664 \def\bbbl@eqdel{()}
6665 \def\bbbl@eqnum{%
6666   {\normalfont\normalcolor
6667     \expandafter\@firstoftwo\bbbl@eqdel
6668     \theequation
6669     \expandafter\@secondoftwo\bbbl@eqdel}}
6670 \def\bbbl@puteqno#1{\eqno\hbox{#1}}
6671 \def\bbbl@putleqno#1{\leqno\hbox{#1}}
6672 \def\bbbl@eqno@flip#1{% So
6673   \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6674     \eqno
6675     \hb@xt@.01pt{%
6676       \hb@xt@\displaywidth{\hss#1\glet\bbbl@upset\@currentlabel}\hss}%
6677   \else
6678     \leqno\hbox{#1\glet\bbbl@upset\@currentlabel}%
6679   \fi
6680   \bbbl@exp{\def\\\@currentlabel{\[bbbl@upset]}}}}
6681 \def\bbbl@leqno@flip#1{%
6682   \ifdim\predisplaysize=-\maxdimen % For consecutive displays
6683     \leqno
6684     \hb@xt@.01pt{%
6685       \hss\hb@xt@\displaywidth{#1\glet\bbbl@upset\@currentlabel}\hss}}%
6686   \else
6687     \eqno\hbox{#1\glet\bbbl@upset\@currentlabel}%
6688   \fi
6689   \bbbl@exp{\def\\\@currentlabel{\[bbbl@upset]}}}}
6690 %
6691 \AtBeginDocument{%
6692   \ifx\bbbl@noamsmath\relax\else
6693     \ifx\maketag@@@\undefined % Normal equation, eqnarray
6694       \ifnum\bbbl@eqnpos=\tw@
6695         \bbbl@replace\equation{\hb@xt@\linewidth}%
6696           {\hbox bdir\mathdirection to\linewidth}%
6697         \bbbl@carg\bbbl@sreplace{[ ]}{\hb@xt@\linewidth}%
6698           {\hbox bdir\mathdirection to\linewidth}%
6699       \fi
6700     \AddToHook{env/equation/begin}{%
6701       \ifnum\bbbl@thetextdir>\z@
6702         \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%
6703         \let\@eqnnum\bbbl@eqnum
6704         \edef\bbbl@eqnodir{\noexpand\bbbl@textdir{\the\bbbl@thetextdir}}%
6705         \chardef\bbbl@thetextdir\z@
6706         \bbbl@add\normalfont{\bbbl@eqnodir}%
6707         \ifcase\bbbl@eqnpos
6708           \let\bbbl@puteqno\bbbl@eqno@flip
6709         \or
6710           \let\bbbl@puteqno\bbbl@leqno@flip
6711         \fi
6712       \fi}%
6713     \ifnum\bbbl@eqnpos=\tw@\else
6714       \def\endequation{\bbbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6715     \fi
6716     \AddToHook{env/eqnarray/begin}{%
6717       \ifnum\bbbl@thetextdir>\z@
6718         \def\bbbl@mathboxdir{\def\bbbl@insidemath{1}}%

```

```

6719     \edef\bbledqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6720     \chardef\bbledthetextdir\z@
6721     \bbledadd\normalfont{\bbledeqnodir}%
6722     \ifnum\bbledeqnpos=@ne
6723         \def\bbledeqnum{%
6724             \setbox\z@\hbox{\bbledeqnum}%
6725             \hbox to 0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6726     \else
6727         \let\bbledeqnum\bbledeqnum
6728     \fi
6729 \fi}
6730 \else % amstex
6731     \ifnum\bbledeqnpos=\tw@
6732         \bbledexp{% Hack to hide maybe undefined conditionals:
6733             \\bbledsreplace\\multline@crcr{\<iffleqn>\tabskip\z@skip\<fi>\crrc}}%
6734     \fi
6735     \bbledexp{% Hack to hide maybe undefined conditionals:
6736         \chardef\bbledeqnpos=0%
6737         \<iftagsleft@>1\<else>\<iffleqn>2\<fi>\<fi>\relax}%
6738     \ifnum\bbledeqnpos=@ne
6739         \let\bbledams@lap\hbox
6740     \else
6741         \let\bbledams@lap\llap
6742     \fi
6743     \ExplSyntaxOn % Required by \bbledsreplace with \intertext@
6744     \bbledsreplace\intertext@\{normalbaselines}%
6745     {\normalbaselines
6746       \ifx\bbledeqnodir\relax\else\bbledpaddir\@ne\bbledeqnodir\fi}%
6747     \ExplSyntaxOff
6748     \def\bbledams@tagbox#1#2{\#1{\bbledeqnodir#2}}% #1=hbox|@lap|flip
6749     \ifx\bbledams@lap\hbox % leqno
6750         \def\bbledams@flip#1{%
6751             \hbox to 0.01pt{\hss\hbox to\displaywidth{\#1\hss}}}%
6752     \else % eqno
6753         \def\bbledams@flip#1{%
6754             \hbox to 0.01pt{\hbox to\displaywidth{\hss\#1}\hss}}%
6755     \fi
6756     \def\bbledams@preset#1{%
6757         \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6758         \ifnum\bbledthetextdir>\z@
6759             \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6760             \bbledsreplace\textdef@\{\hbox}\{\bbledams@tagbox\hbox}%
6761             \bbledsreplace\maketag@@@\{\hbox}\{\bbledams@tagbox#1}%
6762         \fi}%
6763     \def\bbledams@equation{%
6764         \def\bbledmathboxdir{\def\bbledinsidemath{1}}%
6765         \ifnum\bbledthetextdir>\z@
6766             \edef\bbledeqnodir{\noexpand\bbledtextdir{\the\bbledthetextdir}}%
6767             \chardef\bbledthetextdir\z@
6768             \bbledadd\normalfont{\bbledeqnodir}%
6769             \ifcase\bbledeqnpos
6770                 \def\veqno##1##2{\bbledeqno@flip{##1##2}}%
6771             \or
6772                 \def\veqno##1##2{\bbledleqno@flip{##1##2}}%
6773             \fi
6774         \fi}%
6775     \AddToHook{env/equation/begin}{\bbledams@equation}%
6776     \AddToHook{env/equation*/begin}{\bbledams@equation}%
6777     \AddToHook{env/cases/begin}{\bbledams@preset\bbledams@lap}%
6778     \AddToHook{env/multline/begin}{\bbledams@preset\hbox}%
6779     \AddToHook{env/gather/begin}{\bbledams@preset\bbledams@lap}%
6780     \AddToHook{env/gather*/begin}{\bbledams@preset\bbledams@lap}%
6781     \AddToHook{env/align/begin}{\bbledams@preset\bbledams@lap}%

```

```

6782 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6783 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6784 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6785 \AddToHook{env/eqnalign*/begin}{\bbl@ams@preset\hbox}%
6786 % Hackish, for proper alignment. Don't ask me why it works!:
6787 \bbl@exp{% Avoid a 'visible' conditional
6788   \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6789   \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6790 \AddToHook{env/flalign*/begin}{\bbl@ams@preset\hbox}%
6791 \AddToHook{env/split/before}{%
6792   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6793   \ifnum\bbl@thetextdir>\z@
6794     \bbl@ifsamestring\@currentenv{equation}%
6795     {\ifx\bbl@ams@lap\hbox % leqno
6796       \def\bbl@ams@flip#1{%
6797         \hbox to 0.01pt{\hbox to\displaywidth{#1}\hss}\hss}}%
6798     \else
6799       \def\bbl@ams@flip#1{%
6800         \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss#1}}}%
6801     \fi}%
6802   }%
6803   \fi}%
6804 \fi\fi}
6805 \fi

```

Declarations specific to lua, called by \babelprovide.

```

6806 \def\bbl@provide@extra#1{%
6807   % == onchar ==
6808   \ifx\bbl@KVP@onchar\@nnil\else
6809     \bbl@luahyphenate
6810     \bbl@exp{%
6811       \\\AddToHook{env/document/before}{%
6812         {\let\\bbl@ifrestoring\\@firstoftwo
6813           \\\select@language{#1}{}}}%
6814       \directlua{
6815         if Babel.locale_mapped == nil then
6816           Babel.locale_mapped = true
6817           Babel.linebreaking.add_before(Babel.locale_map, 1)
6818           Babel.loc_to_scr = {}
6819           Babel.chr_to_loc = Babel.chr_to_loc or {}
6820         end
6821         Babel.locale_props[\the\localeid].letters = false
6822       }%
6823       \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
6824       \ifin@
6825         \directlua{
6826           Babel.locale_props[\the\localeid].letters = true
6827         }%
6828       \fi
6829       \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
6830       \ifin@
6831         \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
6832           \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
6833         \fi
6834         \bbl@exp{\\\bbl@add\\bbl@starthyphens
6835           {\\\bbl@patterns@lua{\languagename}}}%
6836         \directlua{
6837           if Babel.script_blocks['\bbl@cl{sbc}'] then
6838             Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbc}']
6839             Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
6840           end
6841         }%
6842       \fi

```

```

6843 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
6844 \ifin@
6845 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
6846 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
6847 \directlua{
6848   if Babel.script_blocks['\bbl@cl{sbcpr}'] then
6849     Babel.loc_to_scr[\the\localeid] =
6850     Babel.script_blocks['\bbl@cl{sbcpr}']
6851   end}%
6852 \ifx\bbl@mapselect\@undefined
6853 \AtBeginDocument{%
6854   \bbl@patchfont{\bbl@mapselect}%
6855   {\selectfont}}%
6856 \def\bbl@mapselect{%
6857   \let\bbl@mapselect\relax
6858   \edef\bbl@prefontid{\fontid\font}}%
6859 \def\bbl@mapdir##1{%
6860   \begingroup
6861     \setbox\z@\hbox{% Force text mode
6862       \def\languagename{##1}%
6863       \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
6864       \bbl@switchfont
6865       \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
6866         \directlua{
6867           Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
6868           ['\bbl@prefontid'] = \fontid\font\space}%
6869         \fi}%
6870   \endgroup}%
6871 \fi
6872 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
6873 \fi
6874 \fi
6875 % == mapfont ==
6876 % For bidi texts, to switch the font based on direction. Deprecated
6877 \ifx\bbl@KVP@mapfont\@nnil\else
6878 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
6879 {\bbl@error{unknown-mapfont}}{}{}%
6880 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
6881 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
6882 \ifx\bbl@mapselect\@undefined
6883 \AtBeginDocument{%
6884   \bbl@patchfont{\bbl@mapselect}%
6885   {\selectfont}}%
6886 \def\bbl@mapselect{%
6887   \let\bbl@mapselect\relax
6888   \edef\bbl@prefontid{\fontid\font}}%
6889 \def\bbl@mapdir##1{%
6890   {\def\languagename{##1}%
6891     \let\bbl@ifrestoring\@firstoftwo % avoid font warning
6892     \bbl@switchfont
6893     \directlua{Babel.fontmap
6894       [\the\csname bbl@wdir@##1\endcsname]%
6895       [\bbl@prefontid]=\fontid\font}}}%
6896 \fi
6897 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languagename}}}%
6898 \fi
6899 % == Line breaking: CJK quotes ==
6900 \ifcase\bbl@engine\or
6901 \bbl@xin@{/c}{/\bbl@cl{lbrk}}%
6902 \ifin@
6903 \bbl@ifunset{bbl@quote@\languagename}}{}%
6904 {\directlua{
6905   Babel.locale_props[\the\localeid].cjk_quotes = {}

```

```

6906         local cs = 'op'
6907         for c in string.utfvalues(%
6908             [[\csname bbl@quote@\languagename\endcsname]]) do
6909             if Babel.cjk_characters[c].c == 'qu' then
6910                 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
6911             end
6912             cs = ( cs == 'op') and 'cl' or 'op'
6913         end
6914     }}%
6915 \fi
6916 \fi
6917 % == Counters: mapdigits ==
6918 % Native digits
6919 \ifx\bbl@KVP@mapdigits\@nnil\else
6920 \bbl@ifunset{bbl@dgnat@\languagename}{}%
6921     {\bbl@activate@preotf
6922         \directlua{
6923             Babel.digits_mapped = true
6924             Babel.digits = Babel.digits or {}
6925             Babel.digits[\the\localeid] =
6926                 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6927             if not Babel.numbers then
6928                 function Babel.numbers(head)
6929                     local LOCALE = Babel.attr_locale
6930                     local GLYPH = node.id'glyph'
6931                     local inmath = false
6932                     for item in node.traverse(head) do
6933                         if not inmath and item.id == GLYPH then
6934                             local temp = node.get_attribute(item, LOCALE)
6935                             if Babel.digits[temp] then
6936                                 local chr = item.char
6937                                 if chr > 47 and chr < 58 then
6938                                     item.char = Babel.digits[temp][chr-47]
6939                                 end
6940                             end
6941                         elseif item.id == node.id'math' then
6942                             inmath = (item.subtype == 0)
6943                         end
6944                     end
6945                     return head
6946                 end
6947             end
6948         }}%
6949 \fi
6950 % == transforms ==
6951 \ifx\bbl@KVP@transforms\@nnil\else
6952 \def\bbl@elt##1##2##3{%
6953     \in@{$transforms.}{##1}%
6954     \ifin@
6955         \def\bbl@tempa{##1}%
6956         \bbl@replace\bbl@tempa{transforms.}{}%
6957         \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6958     \fi}%
6959 \bbl@exp{%
6960     \\bbl@ifblank{\bbl@cl{dgnat}}%
6961     {\let\\bbl@tempa\relax}%
6962     {\def\\bbl@tempa{%
6963         \\bbl@elt{transforms.prehyphenation}%
6964         {digits.native.1.0}{([0-9])}%
6965         \\bbl@elt{transforms.prehyphenation}%
6966         {digits.native.1.1}{string={\string|0123456789\string|\bbl@cl{dgnat}}}}}%
6967 \ifx\bbl@tempa\relax\else
6968     \toks@{\expandafter\expandafter\expandafter}%

```

```

6969     \csname bbl@inidata@\languagename\endcsname}%
6970     \bbl@csarg\edef{inidata@\languagename}{%
6971     \unexpanded\expandafter{\bbl@tempa}%
6972     \the\toks@}%
6973     \fi
6974     \csname bbl@inidata@\languagename\endcsname
6975     \bbl@release@transforms\relax % \relax closes the last item.
6976     \fi}

```

Start tabular here:

```

6977 \def\localerestoredirs{%
6978 \ifcase\bbl@thetextdir
6979 \ifnum\textdirection=\z@\else\textdirection=\z@\fi
6980 \else
6981 \ifnum\textdirection=\@ne\else\textdirection=\@ne\fi
6982 \fi
6983 \ifcase\bbl@thepardir
6984 \ifnum\pardirection=\z@\else\pardirection=\z@\bodydirection=\z@\fi
6985 \else
6986 \ifnum\pardirection=\@ne\else\pardirection=\@ne\bodydirection=\@ne\fi
6987 \fi}
6988 %
6989 \IfBabelLayout{tabular}%
6990 {\chardef\bbl@tabular@mode\tw@}% All RTL
6991 {\IfBabelLayout{notabular}%
6992 {\chardef\bbl@tabular@mode\z@}%
6993 {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6994 %
6995 \ifnum\bbl@bidimode>\@ne % Any lua bidi= except default=1
6996 % Redefine: vrules mess up dirs (why?).
6997 \AtBeginDocument{\def\arstrut{\relax\copy\arstrutbox}}%
6998 \ifcase\bbl@tabular@mode\or % 1 = Mixed - default
6999 \let\bbl@parabefore\relax
7000 \AddToHook{para/before}{\bbl@parabefore}
7001 \AtBeginDocument{%
7002 \bbl@replace@tabular{$}{$%
7003 \def\bbl@insidemath{0}%
7004 \def\bbl@parabefore{\localerestoredirs}}%
7005 \ifnum\bbl@tabular@mode=\@ne
7006 \bbl@ifunset{@tabclassz}{}{%
7007 \bbl@exp{% Hide conditionals
7008 \\\bbl@sreplace\\\@tabclassz
7009 {\<ifcase>\\\@chnum}%
7010 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
7011 \@ifpackageloaded{colortbl}%
7012 {\bbl@sreplace\@classz
7013 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
7014 {\@ifpackageloaded{array}%
7015 {\bbl@exp{% Hide conditionals
7016 \\\bbl@sreplace\\\@classz
7017 {\<ifcase>\\\@chnum}%
7018 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
7019 \\\bbl@sreplace\\\@classz
7020 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
7021 {}}%
7022 \fi}%
7023 \or % 2 = All RTL - tabular
7024 \let\bbl@parabefore\relax
7025 \AddToHook{para/before}{\bbl@parabefore}%
7026 \AtBeginDocument{%
7027 \@ifpackageloaded{colortbl}%
7028 {\bbl@replace@tabular{$}{$%
7029 \def\bbl@insidemath{0}%

```



```

7030         \def\bbl@parabefore{\localerestoredirs}}%
7031         \bbl@sreplace@classz
7032         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
7033         {}}%
7034 \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.

```

7035 \AtBeginDocument{%
7036   \@ifpackageloaded{multicol}%
7037     {\toks\expandafter{\multi@column@out}%
7038      \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
7039     {}%
7040   \@ifpackageloaded{paracol}%
7041     {\edef\pcol@output{%
7042      \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
7043     {}}%
7044 \fi

```

Finish here if there in no layout.

```
7045 \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`.

```

7046 \ifnum\bbl@bidimode>\z@ % Any bidi=
7047   \def\bbl@nextfake#1% non-local changes, use always inside a group!
7048     \bbl@exp{%
7049       \mathdir\the\bodydir
7050       #1% Once entered in math, set boxes to restore values
7051       \def\\bbl@insidemath{0}%
7052       \<ifmmode>%
7053         \everyvbox{%
7054           \the\everyvbox
7055           \bodydir\the\bodydir
7056           \mathdir\the\mathdir
7057           \everyhbox{\the\everyhbox}%
7058           \everyvbox{\the\everyvbox}}%
7059         \everyhbox{%
7060           \the\everyhbox
7061           \bodydir\the\bodydir
7062           \mathdir\the\mathdir
7063           \everyhbox{\the\everyhbox}%
7064           \everyvbox{\the\everyvbox}}%
7065         \<fi>}}%
7066 \IfBabelLayout{nopars}
7067 {}
7068 {\edef\bbl@opt@layout{\bbl@opt@layout.pars.}}%
7069 \IfBabelLayout{pars}
7070 {\def\@hangfrom#1{%
7071   \setbox\@tempboxa\hbox{#1}}%
7072   \hangindent\wd\@tempboxa
7073   \ifnum\pagedirection=\pardirection\else
7074     \shapemode\@ne
7075   \fi
7076   \noindent\box\@tempboxa}}
7077 {}
7078 \fi
7079 %
7080 \IfBabelLayout{tabular}
7081 {\let\bbl@0L@tabular\@tabular

```

```

7082 \bbl@replace@tabular{$}{\bbl@nextfake$}%
7083 \let\bbl@NL@@tabular\@tabular
7084 \AtBeginDocument{%
7085   \ifx\bbl@NL@@tabular\@tabular\else
7086     \bbl@exp{\in{\bbl@nextfake}{\@tabular}}%
7087     \ifin\else
7088       \bbl@replace@tabular{$}{\bbl@nextfake$}%
7089       \fi
7090     \let\bbl@NL@@tabular\@tabular
7091     \fi}}
7092 {}

```

We need to patch lists in documents with both LTR and RTL paragraphs. See issue #395 in GitHub. There was a partial solution, but a better one has been devised by Udi Fogiel (in 26.4).

```

7093 \IfBabelLayout{lists}
7094 {\AtBeginDocument{\chardef\bbl@vbox@bodydir\pagedirection}%
7095 \def\bbl@vbox@lists{\chardef\bbl@vbox@bodydir\bodydirection}%
7096 \let\bbl@bidi@vbox\everyvbox
7097 \@nameuse{newtoks}\everyvbox % \outer in Plain
7098 \everyvbox\expandafter{\the\bbl@bidi@vbox}%
7099 \bbl@bidi@vbox{\bbl@vbox@lists\the\everyvbox}%
7100 \let\bbl@OL@list\list
7101 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
7102 \let\bbl@NL@list\list
7103 \def\bbl@listparshape#1#2#3{%
7104   \parshape #1 #2 #3 %
7105   \ifnum\bbl@vbox@bodydir=\pardirection\else
7106     \shapemode\tw@
7107   \fi}}
7108 {}
7109%
7110 \IfBabelLayout{graphics}
7111 {\let\bbl@pictresetdir\relax
7112 \def\bbl@pictsetdir#1{%
7113   \ifcase\bbl@thetextdir
7114     \let\bbl@pictresetdir\relax
7115   \else
7116     \ifcase#1\bodydir TLT % Remember this sets the inner boxes
7117       \or\textdir TLT
7118       \else\bodydir TLT \textdir TLT
7119     \fi
7120     % \(\text|par)dir required in pgf:
7121     \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
7122   \fi}%
7123 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
7124 \directlua{
7125   Babel.get_picture_dir = true
7126   Babel.picture_has_bidi = 0
7127   %
7128   function Babel.picture_dir (head)
7129     if not Babel.get_picture_dir then return head end
7130     if Babel.hlist_has_bidi(head) then
7131       Babel.picture_has_bidi = 1
7132     end
7133     return head
7134   end
7135   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
7136     "Babel.picture_dir")
7137 }%
7138 \AtBeginDocument{%
7139 \def\LS@rot{%
7140   \setbox\@outputbox\ vbox{%
7141     \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%

```

```

7142 \long\def\put(#1,#2)#3{%
7143 \killglue
7144 % Try:
7145 \ifx\bb@pictresetdir\relax
7146 \def\bb@tempc{0}%
7147 \else
7148 \directlua{
7149 Babel.get_picture_dir = true
7150 Babel.picture_has_bidi = 0
7151 }%
7152 \setbox\z@\hb@xt@z@{%
7153 \@defaultunitsset\@tempdimc{#1}\unitlength
7154 \kern\@tempdimc
7155 #3\hss}%
7156 \edef\bb@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
7157 \fi
7158 % Do:
7159 \@defaultunitsset\@tempdimc{#2}\unitlength
7160 \raise\@tempdimc\hb@xt@z@{%
7161 \@defaultunitsset\@tempdimc{#1}\unitlength
7162 \kern\@tempdimc
7163 {\ifnum\bb@tempc>z@\bb@pictresetdir\fi#3}\hss}%
7164 \ignorespaces}%
7165 \MakeRobust\put}%
7166 \AtBeginDocument
7167 {\AddToHook{cmd/diagbox@pict/before}{\let\bb@pictsetdir\@gobble}%
7168 \ifx\pgfpicture\undefined\else
7169 \AddToHook{env/pgfpicture/begin}{\bb@pictsetdir\@ne}%
7170 \bb@add\pgfinterruptpicture{\bb@pictresetdir}%
7171 \bb@add\pgfsys@beginpicture{\bb@pictsetdir\z@}%
7172 \fi
7173 \ifx\tikzpicture\undefined\else
7174 \AddToHook{env/tikzpicture/begin}{\bb@pictsetdir\tw@}%
7175 \bb@add\tikz@atbegin@node{\bb@pictresetdir}%
7176 \bb@replace\tikz{\begingroup}{\begingroup\bb@pictsetdir\tw@}%
7177 \bb@replace\tikzpicture{\begingroup}{\begingroup\bb@pictsetdir\tw@}%
7178 \fi
7179 }}
7180 {}

```

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.

```

7181 \IfBabelLayout{counters*}%
7182 {\bb@add\bb@opt@layout{.counters.}%
7183 \directlua{
7184 luatexbase.add_to_callback("process_output_buffer",
7185 Babel.discard_sublr , "Babel.discard_sublr") }%
7186 }{}
7187 \IfBabelLayout{counters}%
7188 {\let\bb@0L@@textsuperscript\@textsuperscript
7189 \bb@replace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
7190 \let\bb@latinarabic=\@arabic
7191 \let\bb@0L@@arabic\@arabic
7192 \def\@arabic#1{\babelsublr{\bb@latinarabic#1}}%
7193 \ifpackagewith{babel}{bidi=default}%
7194 {\let\bb@asciroman=\@roman
7195 \let\bb@0L@@roman\@roman
7196 \def\@roman#1{\babelsublr{\ensureascii{\bb@asciroman#1}}}%
7197 \let\bb@asciiRoman=\@Roman
7198 \let\bb@0L@@roman\@Roman
7199 \def\@Roman#1{\babelsublr{\ensureascii{\bb@asciiRoman#1}}}%
7200 \let\bb@0L@labelenumii\labelenumii

```

```

7201     \def\labelenumii{}\theenumii}%
7202     \let\bbL@OL@p@enumiii\p@enumiii
7203     \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.

```

7204 \IfBabelLayout{extras}%
7205   {\bbL@ncarg\let\bbL@OL@underline{underline }%
7206    \bbL@carg\bbL@sreplace{underline }%
7207     {\$@@@underline}{\bgroup\bbL@nextfake$@@@underline}%
7208    \bbL@carg\bbL@sreplace{underline }%
7209     {\m@th$}{\m@th$\egroup}%
7210    \let\bbL@OL@LaTeX@LaTeXe
7211    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
7212     \if b\expandafter\car\f@series\@nil\boldmath\fi
7213     \babelsublr{%
7214       \LaTeX\kern.15em2\bbL@nextfake$_{\textstyle\varepsilon}$}}}}
7215   {}
7216 \</luatex>

```

10.13 Lua: transforms

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

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

```

7217 <{*transforms}>
7218 Babel.linebreaking.replacements = {}
7219 Babel.linebreaking.replacements[0] = {} -- pre
7220 Babel.linebreaking.replacements[1] = {} -- post
7221
7222 function Babel.tovalue(v)
7223   if type(v) == 'table' then
7224     return Babel.locale_props[v[1]].vars[v[2]] or v[3]
7225   else
7226     return v
7227   end
7228 end
7229
7230 Babel.attr_hboxed = luatexbase.registernumber'bbL@attr@hboxed'
7231
7232 function Babel.set_hboxed(head, gc)
7233   for item in node.traverse(head) do
7234     node.set_attribute(item, Babel.attr_hboxed, 1)
7235   end
7236   return head
7237 end
7238
7239 Babel.fetch_subtext = {}
7240
7241 Babel.ignore_pre_char = function(node)
7242   return (node.lang == Babel.nohyphenation)
7243 end
7244
7245 Babel.show_transforms = false
7246

```

```

7247 -- Merging both functions doesn't seem feasible, because there are too
7248 -- many differences.
7249 Babel.fetch_subtext[0] = function(head)
7250   local word_string = ''
7251   local word_nodes = {}
7252   local lang
7253   local item = head
7254   local inmath = false
7255
7256   while item do
7257
7258     if item.id == 11 then
7259       inmath = (item.subtype == 0)
7260     end
7261
7262     if inmath then
7263       -- pass
7264
7265     elseif item.id == 29 then
7266       local locale = node.get_attribute(item, Babel.attr_locale)
7267
7268       if lang == locale or lang == nil then
7269         lang = lang or locale
7270         if Babel.ignore_pre_char(item) then
7271           word_string = word_string .. Babel.us_char
7272         else
7273           if node.has_attribute(item, Babel.attr_hboxed) then
7274             word_string = word_string .. Babel.us_char
7275           else
7276             word_string = word_string .. unicode.utf8.char(item.char)
7277           end
7278         end
7279         word_nodes[#word_nodes+1] = item
7280       else
7281         break
7282       end
7283
7284     elseif item.id == 12 and item.subtype == 13 then
7285       if node.has_attribute(item, Babel.attr_hboxed) then
7286         word_string = word_string .. Babel.us_char
7287       else
7288         word_string = word_string .. ' '
7289       end
7290       word_nodes[#word_nodes+1] = item
7291
7292       -- Ignore leading unrecognized nodes, too.
7293     elseif word_string ~= '' then
7294       word_string = word_string .. Babel.us_char
7295       word_nodes[#word_nodes+1] = item -- Will be ignored
7296     end
7297
7298     item = item.next
7299   end
7300
7301   -- Here and above we remove some trailing chars but not the
7302   -- corresponding nodes. But they aren't accessed.
7303   if word_string:sub(-1) == ' ' then
7304     word_string = word_string:sub(1,-2)
7305   end
7306   if Babel.show_transforms then texio.write_nl(word_string) end
7307   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7308   return word_string, word_nodes, item, lang
7309 end

```

```

7310
7311 Babel.fetch_subtext[1] = function(head)
7312   local word_string = ''
7313   local word_nodes = {}
7314   local lang
7315   local item = head
7316   local inmath = false
7317
7318   while item do
7319
7320     if item.id == 11 then
7321       inmath = (item.subtype == 0)
7322     end
7323
7324     if inmath then
7325       -- pass
7326
7327     elseif item.id == 29 then
7328       if item.lang == lang or lang == nil then
7329         lang = lang or item.lang
7330         if node.has_attribute(item, Babel.attr_hboxed) then
7331           word_string = word_string .. Babel.us_char
7332         elseif (item.char == 124) or (item.char == 61) then -- not =, not |
7333           word_string = word_string .. Babel.us_char
7334         else
7335           word_string = word_string .. unicode.utf8.char(item.char)
7336         end
7337         word_nodes[#word_nodes+1] = item
7338       else
7339         break
7340       end
7341
7342     elseif item.id == 7 and item.subtype == 2 then
7343       if node.has_attribute(item, Babel.attr_hboxed) then
7344         word_string = word_string .. Babel.us_char
7345       else
7346         word_string = word_string .. '='
7347       end
7348       word_nodes[#word_nodes+1] = item
7349
7350     elseif item.id == 7 and item.subtype == 3 then
7351       if node.has_attribute(item, Babel.attr_hboxed) then
7352         word_string = word_string .. Babel.us_char
7353       else
7354         word_string = word_string .. '|'
7355       end
7356       word_nodes[#word_nodes+1] = item
7357
7358     -- (1) Go to next word if nothing was found, and (2) implicitly
7359     -- remove leading USs.
7360     elseif word_string == '' then
7361       -- pass
7362
7363     -- This is the responsible for splitting by words.
7364     elseif (item.id == 12 and item.subtype == 13) then
7365       break
7366
7367     else
7368       word_string = word_string .. Babel.us_char
7369       word_nodes[#word_nodes+1] = item -- Will be ignored
7370     end
7371
7372     item = item.next

```

```

7373 end
7374 if Babel.show_transforms then texio.write_nl(word_string) end
7375 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7376 return word_string, word_nodes, item, lang
7377 end
7378
7379 function Babel.pre_hyphenate_replace(head)
7380   Babel.hyphenate_replace(head, 0)
7381 end
7382
7383 function Babel.post_hyphenate_replace(head)
7384   Babel.hyphenate_replace(head, 1)
7385 end
7386
7387 Babel.us_char = string.char(31)
7388
7389 function Babel.hyphenate_replace(head, mode)
7390   local u = unicode.utf8
7391   local lbkr = Babel.linebreaking.replacements[mode]
7392   local tovalue = Babel.tovalue
7393
7394   local word_head = head
7395
7396   if Babel.show_transforms then
7397     texio.write_nl('\n==== Showing ' .. (mode == 0 and 'pre' or 'post') .. 'hyphenation ====')
7398   end
7399
7400   while true do -- for each subtext block
7401
7402     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7403
7404     if Babel.debug then
7405       print()
7406       print((mode == 0) and '@@@@<' or '@@@@>', w)
7407     end
7408
7409     if nw == nil and w == '' then break end
7410
7411     if not lang then goto next end
7412     if not lbkr[lang] then goto next end
7413
7414     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7415     -- loops are nested.
7416     for k=1, #lbkr[lang] do
7417       local p = lbkr[lang][k].pattern
7418       local r = lbkr[lang][k].replace
7419       local attr = lbkr[lang][k].attr or -1
7420
7421       if Babel.debug then
7422         print('*****', p, mode)
7423       end
7424
7425       -- This variable is set in some cases below to the first *byte*
7426       -- after the match, either as found by u.match (faster) or the
7427       -- computed position based on sc if w has changed.
7428       local last_match = 0
7429       local step = 0
7430
7431       -- For every match.
7432       while true do
7433         if Babel.debug then
7434           print('====')
7435         end

```

```

7436     local new -- used when inserting and removing nodes
7437     local dummy_node -- used by after
7438
7439     local matches = { u.match(w, p, last_match) }
7440
7441     if #matches < 2 then break end
7442
7443     -- Get and remove empty captures (with ()'s, which return a
7444     -- number with the position), and keep actual captures
7445     -- (from (...)), if any, in matches.
7446     local first = table.remove(matches, 1)
7447     local last = table.remove(matches, #matches)
7448     -- Non re-fetched substrings may contain \31, which separates
7449     -- subsubstrings.
7450     if string.find(w:sub(first, last-1), Babel.us_char) then break end
7451
7452     local save_last = last -- with A()BC()D, points to D
7453
7454     -- Fix offsets, from bytes to unicode. Explained above.
7455     first = u.len(w:sub(1, first-1)) + 1
7456     last = u.len(w:sub(1, last-1)) -- now last points to C
7457
7458     -- This loop stores in a small table the nodes
7459     -- corresponding to the pattern. Used by 'data' to provide a
7460     -- predictable behavior with 'insert' (w_nodes is modified on
7461     -- the fly), and also access to 'remove'd nodes.
7462     local sc = first-1 -- Used below, too
7463     local data_nodes = {}
7464
7465     local enabled = true
7466     for q = 1, last-first+1 do
7467         data_nodes[q] = w_nodes[sc+q]
7468         if enabled
7469             and attr > -1
7470             and not node.has_attribute(data_nodes[q], attr)
7471         then
7472             enabled = false
7473         end
7474     end
7475
7476     -- This loop traverses the matched substring and takes the
7477     -- corresponding action stored in the replacement list.
7478     -- sc = the position in substr nodes / string
7479     -- rc = the replacement table index
7480     local rc = 0
7481
7482     ----- TODO. dummy_node?
7483     while rc < last-first+1 or dummy_node do -- for each replacement
7484         if Babel.debug then
7485             print('.....', rc + 1)
7486         end
7487         sc = sc + 1
7488         rc = rc + 1
7489
7490         if Babel.debug then
7491             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7492             local ss = ''
7493             for itt in node.traverse(head) do
7494                 if itt.id == 29 then
7495                     ss = ss .. unicode.utf8.char(itt.char)
7496                 else
7497                     ss = ss .. '{' .. itt.id .. '}'
7498                 end
7499             end
7500         end
7501     end

```



```

7499         end
7500         print('*****', ss)
7501
7502     end
7503
7504     local crep = r[rc]
7505     local item = w_nodes[sc]
7506     local item_base = item
7507     local placeholder = Babel.us_char
7508     local d
7509
7510     if crep and crep.data then
7511         item_base = data_nodes[crep.data]
7512     end
7513
7514     if crep then
7515         step = crep.step or step
7516     end
7517
7518     if crep and crep.after then
7519         crep.insert = true
7520         if dummy_node then
7521             item = dummy_node
7522         else -- TODO. if there is a node after?
7523             d = node.copy(item_base)
7524             head, item = node.insert_after(head, item, d)
7525             dummy_node = item
7526         end
7527     end
7528
7529     if crep and not crep.after and dummy_node then
7530         node.remove(head, dummy_node)
7531         dummy_node = nil
7532     end
7533
7534     if not enabled then
7535         last_match = save_last
7536         goto next
7537
7538     elseif crep and next(crep) == nil then -- = {}
7539         if step == 0 then
7540             last_match = save_last -- Optimization
7541         else
7542             last_match = utf8.offset(w, sc+step)
7543         end
7544         goto next
7545
7546     elseif crep == nil or crep.remove then
7547         node.remove(head, item)
7548         table.remove(w_nodes, sc)
7549         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7550         sc = sc - 1 -- Nothing has been inserted.
7551         last_match = utf8.offset(w, sc+1+step)
7552         goto next
7553
7554     elseif crep and crep.kashida then
7555         node.set_attribute(item,
7556             Babel.attr_kashida,
7557             crep.kashida)
7558         last_match = utf8.offset(w, sc+1+step)
7559         goto next
7560
7561     elseif crep and crep.string then

```

```

7562     local str = crep.string(matches)
7563     if str == '' then -- Gather with nil
7564         node.remove(head, item)
7565         table.remove(w_nodes, sc)
7566         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7567         sc = sc - 1 -- Nothing has been inserted.
7568     else
7569         local loop_first = true
7570         for s in string.utfvalues(str) do
7571             d = node.copy(item_base)
7572             d.char = s
7573             if loop_first then
7574                 loop_first = false
7575                 head, new = node.insert_before(head, item, d)
7576                 if sc == 1 then
7577                     word_head = head
7578                 end
7579                 w_nodes[sc] = d
7580                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7581             else
7582                 sc = sc + 1
7583                 head, new = node.insert_before(head, item, d)
7584                 table.insert(w_nodes, sc, new)
7585                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7586             end
7587             if Babel.debug then
7588                 print('.....', 'str')
7589                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7590             end
7591         end -- for
7592         node.remove(head, item)
7593     end -- if ''
7594     last_match = utf8.offset(w, sc+1+step)
7595     goto next
7596
7597 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7598     d = node.new(7, 3) -- (disc, regular)
7599     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7600     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7601     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7602     d.attr = item_base.attr
7603     if crep.pre == nil then -- TeXbook p96
7604         d.penalty = tovalue(crep.penalty) or tex.hyphenpenalty
7605     else
7606         d.penalty = tovalue(crep.penalty) or tex.exhyphenpenalty
7607     end
7608     placeholder = '|'
7609     head, new = node.insert_before(head, item, d)
7610
7611 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7612     -- ERROR
7613
7614 elseif crep and crep.penalty then
7615     d = node.new(14, 0) -- (penalty, userpenalty)
7616     d.attr = item_base.attr
7617     d.penalty = tovalue(crep.penalty)
7618     head, new = node.insert_before(head, item, d)
7619
7620 elseif crep and crep.space then
7621     -- 655360 = 10 pt = 10 * 65536 sp
7622     d = node.new(12, 13) -- (glue, spaceskip)
7623     local quad = font.getfont(item_base.font).size or 655360
7624     node.setglue(d, tovalue(crep.space[1]) * quad,

```

```

7625             tovalue(crep.space[2]) * quad,
7626             tovalue(crep.space[3]) * quad)
7627     if mode == 0 then
7628         placeholder = ' '
7629     end
7630     head, new = node.insert_before(head, item, d)
7631
7632 elseif crep and crep.norule then
7633     -- 655360 = 10 pt = 10 * 65536 sp
7634     d = node.new(2, 3)      -- (rule, empty) = \no*rule
7635     local quad = font.getfont(item_base.font).size or 655360
7636     d.width  = tovalue(crep.norule[1]) * quad
7637     d.height = tovalue(crep.norule[2]) * quad
7638     d.depth  = tovalue(crep.norule[3]) * quad
7639     head, new = node.insert_before(head, item, d)
7640
7641 elseif crep and crep.spacefactor then
7642     d = node.new(12, 13)   -- (glue, spaceskip)
7643     local base_font = font.getfont(item_base.font)
7644     node.setglue(d,
7645         tovalue(crep.spacefactor[1]) * base_font.parameters['space'],
7646         tovalue(crep.spacefactor[2]) * base_font.parameters['space_stretch'],
7647         tovalue(crep.spacefactor[3]) * base_font.parameters['space_shrink'])
7648     if mode == 0 then
7649         placeholder = ' '
7650     end
7651     head, new = node.insert_before(head, item, d)
7652
7653 elseif mode == 0 and crep and crep.space then
7654     -- ERROR
7655
7656 elseif crep and crep.kern then
7657     d = node.new(13, 1)   -- (kern, user)
7658     local quad = font.getfont(item_base.font).size or 655360
7659     d.attr = item_base.attr
7660     d.kern = tovalue(crep.kern) * quad
7661     head, new = node.insert_before(head, item, d)
7662
7663 elseif crep and crep.node then
7664     d = node.new(crep.node[1], crep.node[2])
7665     d.attr = item_base.attr
7666     head, new = node.insert_before(head, item, d)
7667
7668 end -- i.e., replacement cases
7669
7670 -- Shared by disc, space(factor), kern, node and penalty.
7671 if sc == 1 then
7672     word_head = head
7673 end
7674 if crep.insert then
7675     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7676     table.insert(w_nodes, sc, new)
7677     last = last + 1
7678 else
7679     w_nodes[sc] = d
7680     node.remove(head, item)
7681     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7682 end
7683
7684 last_match = utf8.offset(w, sc+1+step)
7685
7686 ::next::
7687

```

```

7688     end -- for each replacement
7689
7690     if Babel.show_transforms then texio.write_nl('> ' .. w) end
7691     if Babel.debug then
7692         print('.....', '/')
7693         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7694     end
7695
7696     if dummy_node then
7697         node.remove(head, dummy_node)
7698         dummy_node = nil
7699     end
7700
7701     end -- for match
7702
7703     end -- for patterns
7704
7705     ::next::
7706     word_head = nw
7707 end -- for substring
7708
7709 if Babel.show_transforms then texio.write_nl(string.rep('-', 32) .. '\n') end
7710 return head
7711 end
7712
7713 -- This table stores capture maps, numbered consecutively
7714 Babel.capture_maps = {}
7715
7716 function Babel.esc_hex_to_char(h)
7717     if tex.getcatcode(tonumber(h, 16)) ~= 11 and
7718         tex.getcatcode(tonumber(h, 16)) ~= 12 then
7719         return string.format([[Uchar"%X ]], tonumber(h,16))
7720     else
7721         return unicode.utf8.char(tonumber(h, 16))
7722     end
7723 end
7724
7725 -- The following functions belong to the next macro
7726 function Babel.capture_func(key, cap)
7727     local ret = "[[" .. cap:gsub('{{([0-9])}}', "[ ]..m[%1]..[[[" .. "]"")"
7728     local cnt
7729     local u = unicode.utf8
7730     ret = u.gsub(ret, '{{(%x%x%x%x+)}}', '\x01%\x04')
7731     ret, cnt = ret:gsub('{{([0-9])|([^\+]|(-))}}', Babel.capture_func_map)
7732     ret = u.gsub(ret, '\x01(%x%x%x%x+)\x04', Babel.esc_hex_to_char)
7733     ret = ret:gsub("%[%]%.%", '')
7734     ret = ret:gsub("%.%[%]%", '')
7735     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7736 end
7737
7738 function Babel.capt_map(from, mapno)
7739     return Babel.capture_maps[mapno][from] or from
7740 end
7741
7742 -- Handle the {n|abc|ABC} syntax in captures
7743 function Babel.capture_func_map(capno, from, to)
7744     local u = unicode.utf8
7745     from = u.gsub(from, '\x01(%x%x%x%x+)\x04',
7746         function (n)
7747             return u.char(tonumber(n, 16))
7748         end)
7749     to = u.gsub(to, '\x01(%x%x%x%x+)\x04',
7750         function (n)

```

```

7751         return u.char(tonumber(n, 16))
7752     end)
7753     local froms = {}
7754     for s in string.utfcharacters(from) do
7755         table.insert(froms, s)
7756     end
7757     local cnt = 1
7758     table.insert(Babel.capture_maps, {})
7759     local mlen = table.getn(Babel.capture_maps)
7760     for s in string.utfcharacters(to) do
7761         Babel.capture_maps[mlen][froms[cnt]] = s
7762         cnt = cnt + 1
7763     end
7764     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7765         (mlen) .. ").. " .. "["
7766 end
7767
7768 -- Create/Extend reversed sorted list of kashida weights:
7769 function Babel.capture_kashida(key, wt)
7770     wt = tonumber(wt)
7771     if Babel.kashida_wts then
7772         for p, q in ipairs(Babel.kashida_wts) do
7773             if wt == q then
7774                 break
7775             elseif wt > q then
7776                 table.insert(Babel.kashida_wts, p, wt)
7777                 break
7778             elseif table.getn(Babel.kashida_wts) == p then
7779                 table.insert(Babel.kashida_wts, wt)
7780             end
7781         end
7782     else
7783         Babel.kashida_wts = { wt }
7784     end
7785     return 'kashida = ' .. wt
7786 end
7787
7788 function Babel.capture_node(id, subtype)
7789     local sbt = 0
7790     for k, v in pairs(node.subtypes(id)) do
7791         if v == subtype then sbt = k end
7792     end
7793     return 'node = {' .. node.id(id) .. ', ' .. sbt .. '}'
7794 end
7795
7796 -- Experimental: applies prehyphenation transforms to a string (letters
7797 -- and spaces).
7798 function Babel.string_prehyphenation(str, locale)
7799     local n, head, last, res
7800     head = node.new(8, 0) -- dummy (hack just to start)
7801     last = head
7802     for s in string.utfvalues(str) do
7803         if s == 20 then
7804             n = node.new(12, 0)
7805         else
7806             n = node.new(29, 0)
7807             n.char = s
7808         end
7809         node.set_attribute(n, Babel.attr_locale, locale)
7810         last.next = n
7811         last = n
7812     end
7813     head = Babel.hyphenate_replace(head, 0)

```

```

7814 res = ''
7815 for n in node.traverse(head) do
7816   if n.id == 12 then
7817     res = res .. ' '
7818   elseif n.id == 29 then
7819     res = res .. unicode.utf8.char(n.char)
7820   end
7821 end
7822 tex.print(res)
7823 end
7824 </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.

```

7825 <*/basic-r>
7826 Babel.bidi_enabled = true
7827
7828 require('babel-data-bidi.lua')
7829
7830 local characters = Babel.characters
7831 local ranges = Babel.ranges
7832
7833 local DIR = node.id("dir")
7834
7835 local function dir_mark(head, from, to, outer)
7836   dir = (outer == 'r') and 'TLT' or 'TRT' -- i.e., reverse

```

```

7837 local d = node.new(DIR)
7838 d.dir = '+' .. dir
7839 node.insert_before(head, from, d)
7840 d = node.new(DIR)
7841 d.dir = '-' .. dir
7842 node.insert_after(head, to, d)
7843 end
7844
7845 function Babel.bidi(head, ispar)
7846   local first_n, last_n      -- first and last char with nums
7847   local last_es              -- an auxiliary 'last' used with nums
7848   local first_d, last_d      -- first and last char in L/R block
7849   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):

```

7850   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7851   local strong_lr = (strong == 'l') and 'l' or 'r'
7852   local outer = strong
7853
7854   local new_dir = false
7855   local first_dir = false
7856   local inmath = false
7857
7858   local last_lr
7859
7860   local type_n = ''
7861
7862   for item in node.traverse(head) do
7863
7864     -- three cases: glyph, dir, otherwise
7865     if item.id == node.id'glyph'
7866       or (item.id == 7 and item.subtype == 2) then
7867
7868       local itemchar
7869       if item.id == 7 and item.subtype == 2 then
7870         itemchar = item.replace.char
7871       else
7872         itemchar = item.char
7873       end
7874       local chardata = characters[itemchar]
7875       dir = chardata and chardata.d or nil
7876       if not dir then
7877         for nn, et in ipairs(ranges) do
7878           if itemchar < et[1] then
7879             break
7880           elseif itemchar <= et[2] then
7881             dir = et[3]
7882             break
7883           end
7884         end
7885       end
7886       dir = dir or 'l'
7887       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).

```

7888   if new_dir then
7889     attr_dir = 0
7890     for at in node.traverse(item.attr) do

```

```

7891         if at.number == Babel.attr_dir then
7892             attr_dir = at.value & 0x3
7893         end
7894     end
7895     if attr_dir == 1 then
7896         strong = 'r'
7897     elseif attr_dir == 2 then
7898         strong = 'al'
7899     else
7900         strong = 'l'
7901     end
7902     strong_lr = (strong == 'l') and 'l' or 'r'
7903     outer = strong_lr
7904     new_dir = false
7905 end
7906
7907 if dir == 'nsm' then dir = strong end          -- W1

```

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

```

7908     dir_real = dir          -- We need dir_real to set strong below
7909     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:

```

7910     if strong == 'al' then
7911         if dir == 'en' then dir = 'an' end          -- W2
7912         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7913         strong_lr = 'r'                             -- W3
7914     end

```

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

```

7915     elseif item.id == node.id'dir' and not inmath then
7916         new_dir = true
7917         dir = nil
7918     elseif item.id == node.id'math' then
7919         inmath = (item.subtype == 0)
7920     else
7921         dir = nil          -- Not a char
7922     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>`.

```

7923     if dir == 'en' or dir == 'an' or dir == 'et' then
7924         if dir ~= 'et' then
7925             type_n = dir
7926         end
7927         first_n = first_n or item
7928         last_n = last_es or item
7929         last_es = nil
7930     elseif dir == 'es' and last_n then -- W3+W6
7931         last_es = item
7932     elseif dir == 'cs' then          -- it's right - do nothing
7933     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7934         if strong_lr == 'r' and type_n ~= '' then
7935             dir_mark(head, first_n, last_n, 'r')
7936         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7937             dir_mark(head, first_n, last_n, 'r')
7938             dir_mark(head, first_d, last_d, outer)
7939             first_d, last_d = nil, nil
7940         elseif strong_lr == 'l' and type_n ~= '' then
7941             last_d = last_n

```



```

7942     end
7943     type_n = ''
7944     first_n, last_n = nil, nil
7945 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:

```

7946     if dir == 'l' or dir == 'r' then
7947         if dir ~= outer then
7948             first_d = first_d or item
7949             last_d = item
7950         elseif first_d and dir ~= strong_lr then
7951             dir_mark(head, first_d, last_d, outer)
7952             first_d, last_d = nil, nil
7953         end
7954     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>`, resp'tly, 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.

```

7955     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7956         item.char = characters[item.char] and
7957             characters[item.char].m or item.char
7958     elseif (dir or new_dir) and last_lr ~= item then
7959         local mir = outer .. strong_lr .. (dir or outer)
7960         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7961             for ch in node.traverse(node.next(last_lr)) do
7962                 if ch == item then break end
7963                 if ch.id == node.id'glyph' and characters[ch.char] then
7964                     ch.char = characters[ch.char].m or ch.char
7965                 end
7966             end
7967         end
7968     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`).

```

7969     if dir == 'l' or dir == 'r' then
7970         last_lr = item
7971         strong = dir_real -- Don't search back - best save now
7972         strong_lr = (strong == 'l') and 'l' or 'r'
7973     elseif new_dir then
7974         last_lr = nil
7975     end
7976 end

```

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

```

7977     if last_lr and outer == 'r' then
7978         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7979             if characters[ch.char] then
7980                 ch.char = characters[ch.char].m or ch.char
7981             end
7982         end
7983     end
7984     if first_n then
7985         dir_mark(head, first_n, last_n, outer)
7986     end
7987     if first_d then
7988         dir_mark(head, first_d, last_d, outer)
7989     end

```

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

```
7990 return node.prev(head) or head
7991 end
7992 </basic-r>
```

And here the Lua code for bidi=basic:

```
7993 <*basic>
7994 -- e.g., Babel.fontmap[1][<prefontid>]=<dirfontid>
7995
7996 Babel.fontmap = Babel.fontmap or {}
7997 Babel.fontmap[0] = {} -- l
7998 Babel.fontmap[1] = {} -- r
7999 Babel.fontmap[2] = {} -- al/an
8000
8001 -- To cancel mirroring. Also OML, OMS, U?
8002 Babel.symbol_fonts = Babel.symbol_fonts or {}
8003 Babel.symbol_fonts[font.id('tenln')] = true
8004 Babel.symbol_fonts[font.id('tenlnw')] = true
8005 Babel.symbol_fonts[font.id('tencirc')] = true
8006 Babel.symbol_fonts[font.id('tencircw')] = true
8007
8008 Babel.bidi_enabled = true
8009 Babel.mirroring_enabled = true
8010
8011 require('babel-data-bidi.lua')
8012
8013 local characters = Babel.characters
8014 local ranges = Babel.ranges
8015
8016 local DIR = node.id('dir')
8017 local GLYPH = node.id('glyph')
8018
8019 local function insert_implicit(head, state, outer)
8020   local new_state = state
8021   if state.sim and state.eim and state.sim ~= state.eim then
8022     dir = ((outer == 'r') and 'TLT' or 'TRT') -- i.e., reverse
8023     local d = node.new(DIR)
8024     d.dir = '+' .. dir
8025     node.insert_before(head, state.sim, d)
8026     local d = node.new(DIR)
8027     d.dir = '-' .. dir
8028     node.insert_after(head, state.eim, d)
8029   end
8030   new_state.sim, new_state.eim = nil, nil
8031   return head, new_state
8032 end
8033
8034 local function insert_numeric(head, state)
8035   local new
8036   local new_state = state
8037   if state.san and state.ean and state.san ~= state.ean then
8038     local d = node.new(DIR)
8039     d.dir = '+TLT'
8040     _, new = node.insert_before(head, state.san, d)
8041     if state.san == state.sim then state.sim = new end
8042     local d = node.new(DIR)
8043     d.dir = '-TLT'
8044     _, new = node.insert_after(head, state.ean, d)
8045     if state.ean == state.eim then state.eim = new end
8046   end
8047   new_state.san, new_state.ean = nil, nil
8048   return head, new_state
```

```

8049 end
8050
8051 local function glyph_not_symbol_font(node)
8052   if node.id == GLYPH then
8053     return not Babel.symbol_fonts[node.font]
8054   else
8055     return false
8056   end
8057 end
8058
8059 -- TODO - \hbox with an explicit dir can lead to wrong results
8060 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
8061 -- was made to improve the situation, but the problem is the 3-dir
8062 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
8063 -- well.
8064
8065 function Babel.bidi(head, ispar, hdir)
8066   local d    -- d is used mainly for computations in a loop
8067   local prev_d = ''
8068   local new_d = false
8069
8070   local nodes = {}
8071   local outer_first = nil
8072   local inmath = false
8073
8074   local glue_d = nil
8075   local glue_i = nil
8076
8077   local has_en = false
8078   local first_et = nil
8079
8080   local has_hyperlink = false
8081
8082   local ATDIR = Babel.attr_dir
8083   local attr_d, temp
8084   local locale_d
8085
8086   local save_outer
8087   local locale_d = node.get_attribute(head, ATDIR)
8088   if locale_d then
8089     locale_d = locale_d & 0x3
8090     save_outer = (locale_d == 0 and 'l') or
8091                 (locale_d == 1 and 'r') or
8092                 (locale_d == 2 and 'al')
8093   elseif ispar then    -- Or error? Shouldn't happen
8094     -- when the callback is called, we are just _after_ the box,
8095     -- and the textdir is that of the surrounding text
8096     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
8097   else
8098     -- Empty box
8099     save_outer = ('TRT' == hdir) and 'r' or 'l'
8100   end
8101   local outer = save_outer
8102   local last = outer
8103   -- 'al' is only taken into account in the first, current loop
8104   if save_outer == 'al' then save_outer = 'r' end
8105
8106   local fontmap = Babel.fontmap
8107
8108   for item in node.traverse(head) do
8109     -- Mask: DxxxPPTT (Done, Pardir [0-2], Textdir [0-2])
8110     locale_d = node.get_attribute(item, ATDIR)
8111     node.set_attribute(item, ATDIR, 0x80)

```

```

8112
8113 -- In what follows, #node is the last (previous) node, because the
8114 -- current one is not added until we start processing the neutrals.
8115 -- three cases: glyph, dir, otherwise
8116 if glyph_not_symbol_font(item)
8117   or (item.id == 7 and item.subtype == 2) then
8118
8119   if locale_d == 0x80 then goto nextnode end
8120   locale_d = locale_d or ((save_outer=='l') and 0 or 1)
8121
8122   local d_font = nil
8123   local item_r
8124   if item.id == 7 and item.subtype == 2 then
8125     item_r = item.replace -- automatic discs have just 1 glyph
8126   else
8127     item_r = item
8128   end
8129
8130   local chardata = characters[item_r.char]
8131   d = chardata and chardata.d or nil
8132   if not d or d == 'nsm' then
8133     for nn, et in ipairs(ranges) do
8134       if item_r.char < et[1] then
8135         break
8136       elseif item_r.char <= et[2] then
8137         if not d then d = et[3]
8138           elseif d == 'nsm' then d_font = et[3]
8139         end
8140         break
8141       end
8142     end
8143   end
8144   d = d or 'l'
8145
8146   -- A short 'pause' in bidi for mapfont
8147   -- %%% TODO. move if fontmap here
8148   d_font = d_font or d
8149   d_font = (d_font == 'l' and 0) or
8150     (d_font == 'nsm' and 0) or
8151     (d_font == 'r' and 1) or
8152     (d_font == 'al' and 2) or
8153     (d_font == 'an' and 2) or nil
8154   if d_font and fontmap and fontmap[d_font][item_r.font] then
8155     item_r.font = fontmap[d_font][item_r.font]
8156   end
8157
8158   if new_d then
8159     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8160     if inmath then
8161       attr_d = 0
8162     else
8163       attr_d = locale_d & 0x3
8164     end
8165     if attr_d == 1 then
8166       outer_first = 'r'
8167       last = 'r'
8168     elseif attr_d == 2 then
8169       outer_first = 'r'
8170       last = 'al'
8171     else
8172       outer_first = 'l'
8173       last = 'l'
8174     end

```

```

8175     outer = last
8176     has_en = false
8177     first_et = nil
8178     new_d = false
8179 end
8180
8181 if glue_d then
8182   if (d == 'l' and 'l' or 'r') ~= glue_d then
8183     table.insert(nodes, {glue_i, 'on', nil})
8184   end
8185   glue_d = nil
8186   glue_i = nil
8187 end
8188
8189 elseif item.id == DIR then
8190   d = nil
8191   new_d = true
8192
8193 elseif item.id == node.id'glue' and item.subtype == 13 then
8194   glue_d = d
8195   glue_i = item
8196   d = nil
8197
8198 elseif item.id == node.id'math' then
8199   inmath = (item.subtype == 0)
8200
8201 elseif item.id == 8 and item.subtype == 19 then
8202   has_hyperlink = true
8203
8204 else
8205   d = nil
8206 end
8207
8208 -- AL <= EN/ET/ES      -- W2 + W3 + W6
8209 if last == 'al' and d == 'en' then
8210   d = 'an'           -- W3
8211 elseif last == 'al' and (d == 'et' or d == 'es') then
8212   d = 'on'           -- W6
8213 end
8214
8215 -- EN + CS/ES + EN     -- W4
8216 if d == 'en' and #nodes >= 2 then
8217   if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
8218     and nodes[#nodes-1][2] == 'en' then
8219     nodes[#nodes][2] = 'en'
8220   end
8221 end
8222
8223 -- AN + CS + AN       -- W4 too, because uax9 mixes both cases
8224 if d == 'an' and #nodes >= 2 then
8225   if (nodes[#nodes][2] == 'cs')
8226     and nodes[#nodes-1][2] == 'an' then
8227     nodes[#nodes][2] = 'an'
8228   end
8229 end
8230
8231 -- ET/EN              -- W5 + W7->l / W6->on
8232 if d == 'et' then
8233   first_et = first_et or (#nodes + 1)
8234 elseif d == 'en' then
8235   has_en = true
8236   first_et = first_et or (#nodes + 1)
8237 elseif first_et then -- d may be nil here !

```

```

8238     if has_en then
8239         if last == 'l' then
8240             temp = 'l'    -- W7
8241         else
8242             temp = 'en'  -- W5
8243         end
8244     else
8245         temp = 'on'     -- W6
8246     end
8247     for e = first_et, #nodes do
8248         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8249     end
8250     first_et = nil
8251     has_en = false
8252 end
8253
8254 -- Force mathdir in math if ON (currently works as expected only
8255 -- with 'l')
8256
8257 if inmath and d == 'on' then
8258     d = ('TRT' == tex.mathdir) and 'r' or 'l'
8259 end
8260
8261 if d then
8262     if d == 'al' then
8263         d = 'r'
8264         last = 'al'
8265     elseif d == 'l' or d == 'r' then
8266         last = d
8267     end
8268     prev_d = d
8269     table.insert(nodes, {item, d, outer_first})
8270 end
8271
8272 outer_first = nil
8273
8274 ::nextnode::
8275
8276 end -- for each node
8277
8278 -- TODO -- repeated here in case EN/ET is the last node. Find a
8279 -- better way of doing things:
8280 if first_et then -- dir may be nil here !
8281     if has_en then
8282         if last == 'l' then
8283             temp = 'l'    -- W7
8284         else
8285             temp = 'en'  -- W5
8286         end
8287     else
8288         temp = 'on'     -- W6
8289     end
8290     for e = first_et, #nodes do
8291         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
8292     end
8293 end
8294
8295 -- dummy node, to close things
8296 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
8297
8298 ----- NEUTRAL -----
8299
8300 outer = save_outer

```

```

8301 last = outer
8302
8303 local first_on = nil
8304
8305 for q = 1, #nodes do
8306     local item
8307
8308     local outer_first = nodes[q][3]
8309     outer = outer_first or outer
8310     last = outer_first or last
8311
8312     local d = nodes[q][2]
8313     if d == 'an' or d == 'en' then d = 'r' end
8314     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
8315
8316     if d == 'on' then
8317         first_on = first_on or q
8318     elseif first_on then
8319         if last == d then
8320             temp = d
8321         else
8322             temp = outer
8323         end
8324         for r = first_on, q - 1 do
8325             nodes[r][2] = temp
8326             item = nodes[r][1] -- MIRRORING
8327             if Babel.mirroring_enabled and glyph_not_symbol_font(item)
8328                 and temp == 'r' and characters[item.char] then
8329                 local font_mode = ''
8330                 if item.font > 0 and font.fonts[item.font].properties then
8331                     font_mode = font.fonts[item.font].properties.mode
8332                 end
8333                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
8334                     item.char = characters[item.char].m or item.char
8335                 end
8336             end
8337         end
8338         first_on = nil
8339     end
8340
8341     if d == 'r' or d == 'l' then last = d end
8342 end
8343
8344 ----- IMPLICIT, REORDER -----
8345
8346 outer = save_outer
8347 last = outer
8348
8349 local state = {}
8350 state.has_r = false
8351
8352 for q = 1, #nodes do
8353
8354     local item = nodes[q][1]
8355
8356     outer = nodes[q][3] or outer
8357
8358     local d = nodes[q][2]
8359
8360     if d == 'nsm' then d = last end -- W1
8361     if d == 'en' then d = 'an' end
8362     local isdir = (d == 'r' or d == 'l')
8363

```

```

8364   if outer == 'l' and d == 'an' then
8365       state.san = state.san or item
8366       state.ean = item
8367   elseif state.san then
8368       head, state = insert_numeric(head, state)
8369   end
8370
8371   if outer == 'l' then
8372       if d == 'an' or d == 'r' then      -- im -> implicit
8373           if d == 'r' then state.has_r = true end
8374           state.sim = state.sim or item
8375           state.eim = item
8376       elseif d == 'l' and state.sim and state.has_r then
8377           head, state = insert_implicit(head, state, outer)
8378       elseif d == 'l' then
8379           state.sim, state.eim, state.has_r = nil, nil, false
8380       end
8381   else
8382       if d == 'an' or d == 'l' then
8383           if nodes[q][3] then -- nil except after an explicit dir
8384               state.sim = item -- so we move sim 'inside' the group
8385           else
8386               state.sim = state.sim or item
8387           end
8388           state.eim = item
8389       elseif d == 'r' and state.sim then
8390           head, state = insert_implicit(head, state, outer)
8391       elseif d == 'r' then
8392           state.sim, state.eim = nil, nil
8393       end
8394   end
8395
8396   if isdir then
8397       last = d          -- Don't search back - best save now
8398   elseif d == 'on' and state.san then
8399       state.san = state.san or item
8400       state.ean = item
8401   end
8402
8403 end
8404
8405 head = node.prev(head) or head
8406 % \end{macrocode}
8407 %
8408 % Now direction nodes has been distributed with relation to characters
8409 % and spaces, we need to take into account \TeX-specific elements in
8410 % the node list, to move them at an appropriate place. Firstly, with
8411 % hyperlinks. Secondly, we avoid them between penalties and spaces, so
8412 % that the latter are still discardable.
8413 %
8414 % \begin{macrocode}
8415 --- FIXES ---
8416 if has_hyperlink then
8417     local flag, linking = 0, 0
8418     for item in node.traverse(head) do
8419         if item.id == DIR then
8420             if item.dir == '+TRT' or item.dir == '+TLT' then
8421                 flag = flag + 1
8422             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8423                 flag = flag - 1
8424             end
8425         elseif item.id == 8 and item.subtype == 19 then
8426             linking = flag

```



```

8427     elseif item.id == 8 and item.subtype == 20 then
8428         if linking > 0 then
8429             if item.prev.id == DIR and
8430                 (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8431                 d = node.new(DIR)
8432                 d.dir = item.prev.dir
8433                 node.remove(head, item.prev)
8434                 node.insert_after(head, item, d)
8435             end
8436         end
8437         linking = 0
8438     end
8439 end
8440 end
8441
8442 for item in node.traverse_id(10, head) do
8443     local p = item
8444     local flag = false
8445     while p.prev and p.prev.id == 14 do
8446         flag = true
8447         p = p.prev
8448     end
8449     if flag then
8450         node.insert_before(head, p, node.copy(item))
8451         node.remove(head, item)
8452     end
8453 end
8454
8455 return head
8456 end
8457
8457 function Babel.unset_atdir(head)
8458     local ATDIR = Babel.attr_dir
8459     for item in node.traverse(head) do
8460         node.set_attribute(item, ATDIR, 0x80)
8461     end
8462     return head
8463 end
8464 (</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.

```

8465 (< *nil>

```

```
8466 \ProvidesLanguage{nil}[<@date@> v<@version@> Nil language]
8467 \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.

```
8468 \ifx\l@nil\undefined
8469 \newlanguage\l@nil
8470 \@namedef{bbl@hyphendata@the\l@nil}{}% Remove warning
8471 \let\bbl@elt\relax
8472 \edef\bbl@languages{% Add it to the list of languages
8473   \bbl@languages\bbl@elt{nil}{the\l@nil}{}%
8474 \fi
```

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

```
8475 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil

\datenil

```
8476 \let\captionnil\@empty
8477 \let\datenil\@empty
```

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

```
8478 \def\bbl@inidata@nil{%
8479   \bbl@elt{identification}{tag.ini}{und}%
8480   \bbl@elt{identification}{load.level}{0}%
8481   \bbl@elt{identification}{charset}{utf8}%
8482   \bbl@elt{identification}{version}{1.0}%
8483   \bbl@elt{identification}{date}{2022-05-16}%
8484   \bbl@elt{identification}{name.local}{nil}%
8485   \bbl@elt{identification}{name.english}{nil}%
8486   \bbl@elt{identification}{name.babel}{nil}%
8487   \bbl@elt{identification}{tag.bcp47}{und}%
8488   \bbl@elt{identification}{language.tag.bcp47}{und}%
8489   \bbl@elt{identification}{tag.opentype}{dflt}%
8490   \bbl@elt{identification}{script.name}{Latin}%
8491   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8492   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8493   \bbl@elt{identification}{level}{1}%
8494   \bbl@elt{identification}{encodings}{}%
8495   \bbl@elt{identification}{derivate}{no}}
8496 \@namedef{bbl@tbc@nil}{und}
8497 \@namedef{bbl@lbc@nil}{und}
8498 \@namedef{bbl@casing@nil}{und}
8499 \@namedef{bbl@lotf@nil}{dflt}
8500 \@namedef{bbl@elname@nil}{nil}
8501 \@namedef{bbl@lname@nil}{nil}
8502 \@namedef{bbl@esname@nil}{Latin}
8503 \@namedef{bbl@sname@nil}{Latin}
8504 \@namedef{bbl@sbc@nil}{Latn}
8505 \@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.

```
8506 \ldf@finish{nil}
8507 </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.

```

8508 << *Compute Julian day >> ≡
8509 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8510 \def\bbl@cs@gregleap#1{%
8511 (\bbl@fpmo{#1}{4} == 0) &&
8512 (!( \bbl@fpmo{#1}{100} == 0) && ( \bbl@fpmo{#1}{400} != 0))}
8513 \def\bbl@cs@jd#1#2#3{% year, month, day
8514 \fpeval{ 1721424.5 + (365 * (#1 - 1)) +
8515 floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8516 floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
8517 ((#2 <= 2) ? 0 : ( \bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
8518 <</Compute Julian day >>

```

13.1. Islamic

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

```

8519 < *ca-islamic >
8520 < @Compute Julian day @ >
8521 % == islamic (default)
8522 % Not yet implemented
8523 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

8524 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8525 ((#3 + ceil(29.5 * (#2 - 1)) +
8526 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8527 1948439.5) - 1) }
8528 \namedef{\bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8529 \namedef{\bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8530 \namedef{\bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8531 \namedef{\bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8532 \namedef{\bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8533 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
8534 \edef\bbl@tempa{%
8535 \fpeval{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1 } }%
8536 \edef#5{%
8537 \fpeval{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) } }%
8538 \edef#6{\fpeval{
8539 min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) } }%
8540 \edef#7{\fpeval{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1 } }

```

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

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

```

8541 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8542 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8543 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8544 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8545 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8546 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8547 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8548 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8549 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8550 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8551 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8552 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
8553 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8554 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8555 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8556 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8557 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%

```

```

8558 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8559 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8560 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8561 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8562 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8563 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8564 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8565 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8566 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8567 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8568 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8569 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8570 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8571 65401,65431,65460,65490,65520}
8572 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8573 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8574 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8575 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8576 \ifnum#2>2014 \ifnum#2<2038
8577 \bbl@afterfi\expandafter\@gobble
8578 \fi\fi
8579 {\bbl@error{year-out-range}{2014-2038}{}}%
8580 \edef\bbl@tempd{\fpeval{ % (Julian) day
8581 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8582 \count@\@ne
8583 \bbl@foreach\bbl@cs@umalqura@data{%
8584 \advance\count@\@ne
8585 \ifnum##1>\bbl@tempd\else
8586 \edef\bbl@tempe{\the\count@}%
8587 \edef\bbl@tempb{##1}%
8588 \fi}%
8589 \edef\bbl@templ{\fpeval{ \bbl@tempe + 16260 + 949 }}% month~lunar
8590 \edef\bbl@tempa{\fpeval{ floor((\bbl@templ - 1) / 12) }}% annus
8591 \edef#5{\fpeval{ \bbl@tempa + 1 }}%
8592 \edef#6{\fpeval{ \bbl@templ - (12 * \bbl@tempa) }}%
8593 \edef#7{\fpeval{ \bbl@tempd - \bbl@tempb + 1 }}%
8594 \bbl@add\bbl@precalendar{%
8595 \bbl@replace\bbl@ld@calendar{-civil}{}}%
8596 \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8597 \bbl@replace\bbl@ld@calendar{+}{}}%
8598 \bbl@replace\bbl@ld@calendar{-}{}}%
8599 </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

```

8600 (*ca-hebrew)
8601 \newcount\bbl@cntcommon
8602 \def\bbl@remainder#1#2#3{%
8603 #3=#1\relax
8604 \divide #3 by #2\relax
8605 \multiply #3 by -#2\relax
8606 \advance #3 by #1\relax}%
8607 \newif\ifbbl@divisible
8608 \def\bbl@checkifdivisible#1#2{%
8609 {\countdef\tmp=0
8610 \bbl@remainder{#1}{#2}{\tmp}%
8611 \ifnum \tmp=0
8612 \global\bbl@divisibletrue
8613 \else
8614 \global\bbl@divisiblefalse

```

```

8615 \fi}}
8616 \newif\ifbbl@gregleap
8617 \def\bbl@ifgregleap#1{%
8618 \bbl@checkifdivisible{#1}{4}%
8619 \ifbbl@divisible
8620 \bbl@checkifdivisible{#1}{100}%
8621 \ifbbl@divisible
8622 \bbl@checkifdivisible{#1}{400}%
8623 \ifbbl@divisible
8624 \bbl@gregleaptrue
8625 \else
8626 \bbl@gregleapfalse
8627 \fi
8628 \else
8629 \bbl@gregleaptrue
8630 \fi
8631 \else
8632 \bbl@gregleapfalse
8633 \fi
8634 \ifbbl@gregleap}
8635 \def\bbl@gregdayspriormonths#1#2#3{%
8636 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8637 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8638 \bbl@ifgregleap{#2}%
8639 \ifnum #1 > 2
8640 \advance #3 by 1
8641 \fi
8642 \fi
8643 \global\bbl@cntcommon=#3}%
8644 #3=\bbl@cntcommon}
8645 \def\bbl@gregdaysprioryears#1#2{%
8646 {\countdef\tmpc=4
8647 \countdef\tmpb=2
8648 \tmpb=#1\relax
8649 \advance \tmpb by -1
8650 \tmpc=\tmpb
8651 \multiply \tmpc by 365
8652 #2=\tmpc
8653 \tmpc=\tmpb
8654 \divide \tmpc by 4
8655 \advance #2 by \tmpc
8656 \tmpc=\tmpb
8657 \divide \tmpc by 100
8658 \advance #2 by -\tmpc
8659 \tmpc=\tmpb
8660 \divide \tmpc by 400
8661 \advance #2 by \tmpc
8662 \global\bbl@cntcommon=#2\relax}%
8663 #2=\bbl@cntcommon}
8664 \def\bbl@absfromgreg#1#2#3#4{%
8665 {\countdef\tmpd=0
8666 #4=#1\relax
8667 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8668 \advance #4 by \tmpd
8669 \bbl@gregdaysprioryears{#3}{\tmpd}%
8670 \advance #4 by \tmpd
8671 \global\bbl@cntcommon=#4\relax}%
8672 #4=\bbl@cntcommon}
8673 \newif\ifbbl@hebrleap
8674 \def\bbl@checkleaphebryear#1{%
8675 {\countdef\tmpa=0
8676 \countdef\tmpb=1
8677 \tmpa=#1\relax

```

```

8678 \multiply \tmpa by 7
8679 \advance \tmpa by 1
8680 \bbl@remainder{\tmpa}{19}{\tmpb}%
8681 \ifnum \tmpb < 7
8682 \global\bbl@hebrleaptrue
8683 \else
8684 \global\bbl@hebrleapfalse
8685 \fi}}
8686 \def\bbl@hebrlapsedmonths#1#2{%
8687 {\countdef\tmpa=0
8688 \countdef\tmpb=1
8689 \countdef\tmpc=2
8690 \tmpa=#1\relax
8691 \advance \tmpa by -1
8692 #2=\tmpa
8693 \divide #2 by 19
8694 \multiply #2 by 235
8695 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8696 \tmpc=\tmpb
8697 \multiply \tmpb by 12
8698 \advance #2 by \tmpb
8699 \multiply \tmpc by 7
8700 \advance \tmpc by 1
8701 \divide \tmpc by 19
8702 \advance #2 by \tmpc
8703 \global\bbl@cntcommon=#2}%
8704 #2=\bbl@cntcommon}
8705 \def\bbl@hebrlapseddays#1#2{%
8706 {\countdef\tmpa=0
8707 \countdef\tmpb=1
8708 \countdef\tmpc=2
8709 \bbl@hebrlapsedmonths{#1}{#2}%
8710 \tmpa=#2\relax
8711 \multiply \tmpa by 13753
8712 \advance \tmpa by 5604
8713 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8714 \divide \tmpa by 25920
8715 \multiply #2 by 29
8716 \advance #2 by 1
8717 \advance #2 by \tmpa
8718 \bbl@remainder{#2}{7}{\tmpa}%
8719 \ifnum \tmpc < 19440
8720 \ifnum \tmpc < 9924
8721 \else
8722 \ifnum \tmpa=2
8723 \bbl@checkleaphebrewyear{#1}% of a common year
8724 \ifbbl@hebrleap
8725 \else
8726 \advance #2 by 1
8727 \fi
8728 \fi
8729 \fi
8730 \ifnum \tmpc < 16789
8731 \else
8732 \ifnum \tmpa=1
8733 \advance #1 by -1
8734 \bbl@checkleaphebrewyear{#1}% at the end of leap year
8735 \ifbbl@hebrleap
8736 \advance #2 by 1
8737 \fi
8738 \fi
8739 \fi
8740 \else

```

```

8741     \advance #2 by 1
8742 \fi
8743 \bbl@remainder{#2}{7}{\tmpa}%
8744 \ifnum \tmpa=0
8745     \advance #2 by 1
8746 \else
8747     \ifnum \tmpa=3
8748         \advance #2 by 1
8749     \else
8750         \ifnum \tmpa=5
8751             \advance #2 by 1
8752         \fi
8753     \fi
8754 \fi
8755 \global\bbl@cntcommon=#2\relax}%
8756 #2=\bbl@cntcommon}
8757 \def\bbl@daysinhebrewyear#1#2{%
8758     {\countdef\tmpe=12
8759     \bbl@hebreleapseddays{#1}{\tmpe}%
8760     \advance #1 by 1
8761     \bbl@hebreleapseddays{#1}{#2}%
8762     \advance #2 by -\tmpe
8763     \global\bbl@cntcommon=#2}%
8764 #2=\bbl@cntcommon}
8765 \def\bbl@hebrdayspriormonths#1#2#3{%
8766     {\countdef\tmpf= 14
8767     #3=\ifcase #1
8768         0 \or
8769         0 \or
8770         30 \or
8771         59 \or
8772         89 \or
8773         118 \or
8774         148 \or
8775         148 \or
8776         177 \or
8777         207 \or
8778         236 \or
8779         266 \or
8780         295 \or
8781         325 \or
8782         400
8783     \fi
8784     \bbl@checkleaphebrewyear{#2}%
8785     \ifbbl@hebrleap
8786         \ifnum #1 > 6
8787             \advance #3 by 30
8788         \fi
8789     \fi
8790     \bbl@daysinhebrewyear{#2}{\tmpf}%
8791     \ifnum #1 > 3
8792         \ifnum \tmpf=353
8793             \advance #3 by -1
8794         \fi
8795         \ifnum \tmpf=383
8796             \advance #3 by -1
8797         \fi
8798     \fi
8799     \ifnum #1 > 2
8800         \ifnum \tmpf=355
8801             \advance #3 by 1
8802         \fi
8803     \ifnum \tmpf=385

```

```

8804         \advance #3 by 1
8805     \fi
8806 \fi
8807 \global\bbl@cntcommon=#3\relax}%
8808 #3=\bbl@cntcommon}
8809 \def\bbl@absfromhebr#1#2#3#4{%
8810     {#4=#1\relax
8811     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8812     \advance #4 by #1\relax
8813     \bbl@hebrrelapseddays{#3}{#1}%
8814     \advance #4 by #1\relax
8815     \advance #4 by -1373429
8816     \global\bbl@cntcommon=#4\relax}%
8817 #4=\bbl@cntcommon}
8818 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8819     {\countdef\tmpx= 17
8820     \countdef\tmpy= 18
8821     \countdef\tmpz= 19
8822     #6=#3\relax
8823     \global\advance #6 by 3761
8824     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8825     \tmpz=1 \tmpy=1
8826     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8827     \ifnum \tmpx > #4\relax
8828         \global\advance #6 by -1
8829         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8830     \fi
8831     \advance #4 by -\tmpx
8832     \advance #4 by 1
8833     #5=#4\relax
8834     \divide #5 by 30
8835     \loop
8836         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8837         \ifnum \tmpx < #4\relax
8838             \advance #5 by 1
8839             \tmpy=\tmpx
8840         \repeat
8841     \global\advance #5 by -1
8842     \global\advance #4 by -\tmpy}}
8843 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8844 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8845 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8846     \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8847     \bbl@hebrfromgreg
8848     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8849     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8850 \edef#4{\the\bbl@hebryear}%
8851 \edef#5{\the\bbl@hebrmonth}%
8852 \edef#6{\the\bbl@hebrday}}
8853 </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).

```

8854 <*ca-persian>
8855 <@Compute Julian day@>
8856 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8857     2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8858 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%

```



```

8859 \edef\bbbl@tempa{#1}% 20XX-03-\bbbl@tempe = 1 farvardin:
8860 \ifnum\bbbl@tempa>2012 \ifnum\bbbl@tempa<2051
8861 \bbbl@afterfi\expandafter\@gobble
8862 \fi\fi
8863 {\bbbl@error{year-out-range}{2013-2050}}{}}}%
8864 \bbbl@xin@{\bbbl@tempa}{\bbbl@cs@firstjal@xx}%
8865 \ifin@def\bbbl@tempe{20}\else\def\bbbl@tempe{21}\fi
8866 \edef\bbbl@tempc{\fpeval{\bbbl@cs@jd{\bbbl@tempa}{#2}{#3}+.5}}% current
8867 \edef\bbbl@tempb{\fpeval{\bbbl@cs@jd{\bbbl@tempa}{03}{\bbbl@tempe}+.5}}% begin
8868 \ifnum\bbbl@tempc<\bbbl@tempb
8869 \edef\bbbl@tempa{\fpeval{\bbbl@tempa-1}}% go back 1 year and redo
8870 \bbbl@xin@{\bbbl@tempa}{\bbbl@cs@firstjal@xx}%
8871 \ifin@def\bbbl@tempe{20}\else\def\bbbl@tempe{21}\fi
8872 \edef\bbbl@tempb{\fpeval{\bbbl@cs@jd{\bbbl@tempa}{03}{\bbbl@tempe}+.5}}%
8873 \fi
8874 \edef#4{\fpeval{\bbbl@tempa-621}}% set Jalali year
8875 \edef#6{\fpeval{\bbbl@tempc-\bbbl@tempb+1}}% days from 1 farvardin
8876 \edef#5{\fpeval{% set Jalali month
8877 (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8878 \edef#6{\fpeval{% set Jalali day
8879 (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : ((#5 - 1) * 30) + 6))}}%
8880 </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.

```

8881 (*ca-coptic)
8882 <@Compute Julian day@>
8883 \def\bbbl@ca@coptic#1-#2-#3\@#4#5#6{%
8884 \edef\bbbl@tempd{\fpeval{\floor{\bbbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8885 \edef\bbbl@tempc{\fpeval{\bbbl@tempd - 1825029.5}}%
8886 \edef#4{\fpeval{%
8887 floor((\bbbl@tempc - floor((\bbbl@tempc+366) / 1461)) / 365) + 1}}%
8888 \edef\bbbl@tempc{\fpeval{%
8889 \bbbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8890 \edef#5{\fpeval{\floor{\bbbl@tempc / 30} + 1}}%
8891 \edef#6{\fpeval{\bbbl@tempc - (#5 - 1) * 30 + 1}}%
8892 </ca-coptic>
8893 (*ca-ethiopic)
8894 <@Compute Julian day@>
8895 \def\bbbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8896 \edef\bbbl@tempd{\fpeval{\floor{\bbbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8897 \edef\bbbl@tempc{\fpeval{\bbbl@tempd - 1724220.5}}%
8898 \edef#4{\fpeval{%
8899 floor((\bbbl@tempc - floor((\bbbl@tempc+366) / 1461)) / 365) + 1}}%
8900 \edef\bbbl@tempc{\fpeval{%
8901 \bbbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8902 \edef#5{\fpeval{\floor{\bbbl@tempc / 30} + 1}}%
8903 \edef#6{\fpeval{\bbbl@tempc - (#5 - 1) * 30 + 1}}%
8904 </ca-ethiopic>

```

13.5. Julian

Based on [ReinDersh].

```

8905 (*ca-julian)
8906 <@Compute Julian day@>
8907 \def\bbbl@ca@julian#1-#2-#3\@#4#5#6{%
8908 \edef\bbbl@tempj{\fpeval{\floor{\bbbl@cs@jd{#1}{#2}{#3}) + .5}}%
8909 \edef\bbbl@tempa{\fpeval{\bbbl@tempj + 32082.5}}%
8910 \edef\bbbl@tempb{\fpeval{\floor{(4 * \bbbl@tempa + 3) / 1461)}}%
8911 \edef\bbbl@tempc{\fpeval{\bbbl@tempa - floor(1461*\bbbl@tempb/4)}}%

```

```

8912 \edef\bbl@tempd{\fpeval{\floor((5 * \bbl@tempc + 2) / 153)}}%
8913 \edef#6{\fpeval{\bbl@tempc - floor((153*\bbl@tempd+2) / 5) + 1}}%
8914 \edef#5{\fpeval{\bbl@tempd + 3 - 12 * floor(\bbl@tempd / 10)}}%
8915 \edef#4{\fpeval{\bbl@tempb - 4800 + floor(\bbl@tempd / 10)}}%
8916 </ca-julian>

```

13.6. Buddhist

That's very simple.

```

8917 <*ca-buddhist>
8918 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8919 \edef#4{\number\numexpr#1+543\relax}%
8920 \edef#5{#2}%
8921 \edef#6{#3}}
8922 </ca-buddhist>
8923 %
8924 % \subsection{Chinese}
8925 %
8926 % Brute force, with the Julian day of first day of each month. The
8927 % table has been computed with the help of \textsf{python-lunardate} by
8928 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8929 % is 2015-2044.
8930 %
8931 % \begin{macrocode}
8932 <*ca-chinese>
8933 \ExplSyntaxOn
8934 <@Compute Julian day@>
8935 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8936 \edef\bbl@tempd{\fpeval{%
8937 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8938 \count@z@
8939 \@tempcnta=2015
8940 \bbl@foreach\bbl@cs@chinese@data{%
8941 \ifnum##1>\bbl@tempd\else
8942 \advance\count@\@ne
8943 \ifnum\count@>12
8944 \count@\@ne
8945 \advance\@tempcnta\@ne\fi
8946 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8947 \ifin@
8948 \advance\count@\m@ne
8949 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8950 \else
8951 \edef\bbl@tempe{\the\count@}%
8952 \fi
8953 \edef\bbl@tempb{##1}%
8954 \fi}%
8955 \edef#4{\the\@tempcnta}%
8956 \edef#5{\bbl@tempe}%
8957 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8958 \def\bbl@cs@chinese@leap{%
8959 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8960 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8961 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8962 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8963 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8964 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8965 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8966 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8967 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8968 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8969 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8970 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%

```

```

8971 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8972 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8973 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8974 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8975 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8976 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8977 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8978 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8979 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8980 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8981 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8982 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8983 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8984 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8985 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8986 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8987 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8988 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8989 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8990 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8991 10896,10926,10956,10986,11015,11045,11074,11103}
8992 \ExplSyntaxOff
8993 </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`.

```

8994 (*bplain | blplain)
8995 \catcode`\{=1 % left brace is begin-group character
8996 \catcode`\}=2 % right brace is end-group character
8997 \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).

```

8998 \openin 0 hyphen.cfg
8999 \ifeof0
9000 \else
9001 \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.

```

9002 \def\input #1 {%
9003 \let\input\a
9004 \a hyphen.cfg
9005 \let\a\undefined
9006 }

```

```
9007 \fi
9008 </bplain | bplain>
```

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

```
9009 <bplain>\a plain.tex
9010 <bplain>\a lplain.tex
```

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

```
9011 <bplain>\def\fmtname{babel-plain}
9012 <bplain>\def\fmtname{babel-lplain}
```

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

14.2. Emulating some \LaTeX features

The file `babel.def` expects some definitions made in the $\LaTeX 2_{\epsilon}$ 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).

```
9013 <<*Emulate LaTeX>> ≡
9014 \def\@empty{}
9015 \def\loadlocalcfg#1{%
9016   \openin0#1.cfg
9017   \ifeof0
9018     \closein0
9019   \else
9020     \closein0
9021     {\immediate\write16{*****}%
9022      \immediate\write16{* Local config file #1.cfg used}%
9023      \immediate\write16{*}%
9024     }
9025   \input #1.cfg\relax
9026   \fi
9027   \@endofldf}
```

14.3. General tools

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

```
9028 \long\def\@firstofone#1{#1}
9029 \long\def\@firstoftwo#1#2{#1}
9030 \long\def\@secondoftwo#1#2{#2}
9031 \def\@nnil{\@nil}
9032 \def\@gobbletwo#1#2{}
9033 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
9034 \def\@star@or@long#1{%
9035   \@ifstar
9036   {\let\@ngrel@x\relax#1}%
9037   {\let\@ngrel@x\long#1}}
9038 \let\@ngrel@x\relax
9039 \def\@car#1#2\@nil{#1}
9040 \def\@cdr#1#2\@nil{#2}
9041 \let\@typeset@protect\relax
9042 \let\protected@edef\edef
9043 \long\def\@gobble#1{}
9044 \edef\@backslashchar{\expandafter\@gobble\string\}
9045 \def\strip@prefix#1>{}
9046 \def@g@addto@macro#1#2{{%
9047   \toks@\expandafter{#1#2}%
9048   \xdef#1{the\toks@}}}
```

```

9049 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
9050 \def\@nameuse#1{\csname #1\endcsname}
9051 \def\@ifundefined#1{%
9052   \expandafter\ifx\csname#1\endcsname\relax
9053     \expandafter\@firstoftwo
9054   \else
9055     \expandafter\@secondoftwo
9056   \fi}
9057 \def\@expandtwoargs#1#2#3{%
9058   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
9059 \def\zap@space#1 #2{%
9060   #1%
9061   \ifx#2\@empty\else\expandafter\zap@space\fi
9062   #2}
9063 \let\bbl@trace@gobble
9064 \def\bbl@error#1{% Implicit #2#3#4
9065   \begingroup
9066     \catcode`\=0 \catcode`\==12 \catcode`\`=12
9067     \catcode`\^^M=5 \catcode`\%=14
9068     \input errbabel.def
9069   \endgroup
9070   \bbl@error{#1}}
9071 \def\bbl@warning#1{%
9072   \begingroup
9073     \newlinechar=`^^J
9074     \def\{^^J(babel) }%
9075     \message{\#1}%
9076   \endgroup}
9077 \let\bbl@infowarn\bbl@warning
9078 \def\bbl@info#1{%
9079   \begingroup
9080     \newlinechar=`^^J
9081     \def\{^^J}%
9082     \wlog{#1}%
9083   \endgroup}

```

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

```

9084 \ifx\@preamblecmds\undefined
9085   \def\@preamblecmds{}
9086 \fi
9087 \def\@onlypreamble#1{%
9088   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
9089     \@preamblecmds\do#1}}
9090 \@onlypreamble\@onlypreamble

```

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

```

9091 \def\begindocument{%
9092   \@begindocumenthook
9093   \global\let\@begindocumenthook\@undefined
9094   \def\do##1{\global\let##1\@undefined}%
9095   \@preamblecmds
9096   \global\let\do\noexpand}
9097 \ifx\@begindocumenthook\@undefined
9098   \def\@begindocumenthook{}
9099 \fi
9100 \@onlypreamble\@begindocumenthook
9101 \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`.

```

9102 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
9103 \@onlypreamble\AtEndOfPackage

```

```

9104 \def\@endofldf{}
9105 \@onlypreamble\@endofldf
9106 \let\bbll@afterlang\@empty
9107 \chardef\bbll@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.

```

9108 \catcode`\&=\z@
9109 \ifx&\if@filesw\@undefined
9110 \expandafter\let\csname if@filesw\expandafter\endcsname
9111 \csname iffalse\endcsname
9112 \fi
9113 \catcode`\&=4

```

Mimic \LaTeX 's commands to define control sequences.

```

9114 \def\newcommand{\@star@or@long\new@command}
9115 \def\new@command#1{%
9116 \@testopt{\@newcommand#1}0}
9117 \def\@newcommand#1[#2]{%
9118 \@ifnextchar [{\@xargdef#1[#2]}%
9119 \@argdef#1[#2]}
9120 \long\def\@argdef#1[#2]#3{%
9121 \@yargdef#1\@ne{#2}{#3}}
9122 \long\def\@xargdef#1[#2][#3]#4{%
9123 \expandafter\def\expandafter#1\expandafter{%
9124 \expandafter\@protected@testopt\expandafter #1%
9125 \csname\string#1\expandafter\endcsname{#3}}%
9126 \expandafter\@yargdef \csname\string#1\endcsname
9127 \tw@{#2}{#4}}
9128 \long\def\@yargdef#1#2#3{%
9129 \@tempcnta#3\relax
9130 \advance \@tempcnta \@ne
9131 \let\@hash@\relax
9132 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
9133 \@tempcntb #2%
9134 \@whilenum \@tempcntb <\@tempcnta
9135 \do{%
9136 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
9137 \advance\@tempcntb \@ne}%
9138 \let\@hash@##%
9139 \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
9140 \def\providecommand{\@star@or@long\provide@command}
9141 \def\provide@command#1{%
9142 \begingroup
9143 \escapechar\m@ne\xdef\@gtempa{\string#1}%
9144 \endgroup
9145 \expandafter\@ifundefined\@gtempa
9146 {\def\reserved@a{\new@command#1}}%
9147 {\let\reserved@a\relax
9148 \def\reserved@a{\new@command\reserved@a}}%
9149 \reserved@a}%
9150 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
9151 \def\declare@robustcommand#1{%
9152 \edef\reserved@a{\string#1}%
9153 \def\reserved@b{#1}%
9154 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
9155 \edef#1{%
9156 \ifx\reserved@a\reserved@b
9157 \noexpand\x@protect
9158 \noexpand#1%
9159 \fi
9160 \noexpand\protect

```

```

9161     \expandafter\noexpand\csname
9162     \expandafter\@gobble\string#1 \endcsname
9163 }%
9164 \expandafter\new@command\csname
9165     \expandafter\@gobble\string#1 \endcsname
9166 }
9167 \def\x@protect#1{%
9168     \ifx\protect\@typeset@protect\else
9169         \@x@protect#1%
9170     \fi
9171 }
9172 \catcode`\&=\z@ % Trick to hide conditionals
9173 \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`.

```

9174 \def\bbl@tempa{\csname newif\endcsname&ifin@}
9175 \catcode`\&=4
9176 \ifx\in@\@undefined
9177     \def\in@#1#2{%
9178         \def\in@@##1##2##3\in@@{%
9179             \ifx\in@@##2\in@false\else\in@true\fi}%
9180         \in@@##1\in@\in@@}
9181 \else
9182     \let\bbl@tempa\@empty
9183 \fi
9184 \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).

```

9185 \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.

```

9186 \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.

```

9187 \ifx\@tempcnta\@undefined
9188     \csname newcount\endcsname\@tempcnta\relax
9189 \fi
9190 \ifx\@tempcntb\@undefined
9191     \csname newcount\endcsname\@tempcntb\relax
9192 \fi

```

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

```

9193 \ifx\bye\@undefined
9194     \advance\count10 by -2\relax
9195 \fi
9196 \ifx\@ifnextchar\@undefined
9197     \def\@ifnextchar#1#2#3{%
9198         \let\reserved@d=#1%
9199         \def\reserved@a{#2}\def\reserved@b{#3}%
9200         \futurelet\@let@token\@ifnch}
9201 \def\@ifnch{%
9202     \ifx\@let@token\@sptoken
9203         \let\reserved@c\@xifnch
9204     \else

```

```

9205     \ifx\@let@token\reserved@d
9206     \let\reserved@c\reserved@a
9207     \else
9208     \let\reserved@c\reserved@b
9209     \fi
9210 \fi
9211 \reserved@c}
9212 \def\{\let\@sptoken= } \: % this makes \@sptoken a space token
9213 \def\{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
9214 \fi
9215 \def\@testopt#1#2{%
9216 \@ifnextchar[#{1}{#1[#2]}}
9217 \def\@protected\@testopt#1{%
9218 \ifx\protect\@typeset@protect
9219 \expandafter\@testopt
9220 \else
9221 \@x@protect#1%
9222 \fi}
9223 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
9224 #2\relax}\fi}
9225 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
9226 \else\expandafter\@gobble\fi{#1}}

```

14.4. Encoding related macros

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

```

9227 \def\DeclareTextCommand{%
9228 \@dec@text@cmd\providecommand
9229 }
9230 \def\ProvideTextCommand{%
9231 \@dec@text@cmd\providecommand
9232 }
9233 \def\DeclareTextSymbol#1#2#3{%
9234 \@dec@text@cmd\chardef#1{#2}#3\relax
9235 }
9236 \def\@dec@text@cmd#1#2#3{%
9237 \expandafter\def\expandafter#2%
9238 \expandafter{%
9239 \csname#3-cmd\expandafter\endcsname
9240 \expandafter#2%
9241 \csname#3\string#2\endcsname
9242 }%
9243 % \let\@ifdefinable\@rc@ifdefinable
9244 \expandafter#1\csname#3\string#2\endcsname
9245 }
9246 \def\@current@cmd#1{%
9247 \ifx\protect\@typeset@protect\else
9248 \noexpand#1\expandafter\@gobble
9249 \fi
9250 }
9251 \def\@changed@cmd#1#2{%
9252 \ifx\protect\@typeset@protect
9253 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
9254 \expandafter\ifx\csname ?\string#1\endcsname\relax
9255 \expandafter\def\csname ?\string#1\endcsname{%
9256 \@changed@x@err{#1}%
9257 }%
9258 \fi
9259 \global\expandafter\let
9260 \csname\cf@encoding\string#1\expandafter\endcsname
9261 \csname ?\string#1\endcsname
9262 \fi
9263 \csname\cf@encoding\string#1%

```



```

9264     \expandafter\endcsname
9265 \else
9266     \noexpand#1%
9267 \fi
9268 }
9269 \def\@changed@x@err#1{%
9270     \errhelp{Your command will be ignored, type <return> to proceed}%
9271     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
9272 \def\DeclareTextCommandDefault#1{%
9273     \DeclareTextCommand#1?%
9274 }
9275 \def\ProvideTextCommandDefault#1{%
9276     \ProvideTextCommand#1?%
9277 }
9278 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
9279 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
9280 \def\DeclareTextAccent#1#2#3{%
9281     \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
9282 }
9283 \def\DeclareTextCompositeCommand#1#2#3#4{%
9284     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
9285     \edef\reserved@b{\string##1}%
9286     \edef\reserved@c{%
9287         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
9288     \ifx\reserved@b\reserved@c
9289         \expandafter\expandafter\expandafter\ifx
9290             \expandafter\@car\reserved@a\relax\relax\@nil
9291             \@text@composite
9292     \else
9293         \edef\reserved@b##1{%
9294             \def\expandafter\noexpand
9295                 \csname#2\string#1\endcsname###1{%
9296                 \noexpand\@text@composite
9297                 \expandafter\noexpand\csname#2\string#1\endcsname
9298                 ###1\noexpand\@empty\noexpand\@text@composite
9299                 {##1}%
9300             }%
9301         }%
9302         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
9303     \fi
9304     \expandafter\def\csname\expandafter\string\csname
9305         #2\endcsname\string#1-\string#3\endcsname{#4}
9306 \else
9307     \errhelp{Your command will be ignored, type <return> to proceed}%
9308     \errmessage{\string\DeclareTextCompositeCommand\space used on
9309         inappropriate command \protect#1}
9310 \fi
9311 }
9312 \def\@text@composite#1#2#3\@text@composite{%
9313     \expandafter\@text@composite@x
9314     \csname\string#1-\string#2\endcsname
9315 }
9316 \def\@text@composite@x#1#2{%
9317     \ifx#1\relax
9318         #2%
9319     \else
9320         #1%
9321     \fi
9322 }
9323 %
9324 \def\@strip@args#1:#2-#3\@strip@args{#2}
9325 \def\DeclareTextComposite#1#2#3#4{%
9326     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%

```

```

9327 \bgroup
9328 \lccode` \@=#4%
9329 \lowercase{%
9330 \egroup
9331 \reserved@a @%
9332 }%
9333 }
9334 %
9335 \def\UseTextSymbol#1#2#{#2}
9336 \def\UseTextAccent#1#2#3{
9337 \def\@use@text@encoding#1{
9338 \def\DeclareTextSymbolDefault#1#2{%
9339 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
9340 }
9341 \def\DeclareTextAccentDefault#1#2{%
9342 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
9343 }
9344 \def\cf@encoding{OT1}

```

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

```

9345 \DeclareTextAccent{\"}{OT1}{127}
9346 \DeclareTextAccent{\'}{OT1}{19}
9347 \DeclareTextAccent{\^}{OT1}{94}
9348 \DeclareTextAccent{\`}{OT1}{18}
9349 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

9350 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
9351 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
9352 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
9353 \DeclareTextSymbol{\textquoteright}{OT1}{``}
9354 \DeclareTextSymbol{\i}{OT1}{16}
9355 \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`.

```

9356 \ifx\scriptsize\@undefined
9357 \let\scriptsize\sevenrm
9358 \fi

```

And a few more “dummy” definitions.

```

9359 \def\languagename{english}%
9360 \let\bbl@opt@shorthands\@nnil
9361 \def\bbl@ifshorthand#1#2#3#{#2}%
9362 \let\bbl@language@opts\@empty
9363 \let\bbl@provide@locale\relax
9364 \ifx\babeloptionstrings\@undefined
9365 \let\bbl@opt@strings\@nnil
9366 \else
9367 \let\bbl@opt@strings\babeloptionstrings
9368 \fi
9369 \def\BabelStringsDefault{generic}
9370 \def\bbl@tempa{normal}
9371 \ifx\babeloptionmath\bbl@tempa
9372 \def\bbl@mathnormal{\noexpand\textormath}
9373 \fi
9374 \def\AfterBabelLanguage#1#2{
9375 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
9376 \let\bbl@afterlang\relax
9377 \def\bbl@opt@safe{BR}
9378 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
9379 \ifx\bbl@trace\@undefined\def\bbl@trace#1{\fi
9380 \expandafter\newif\cname ifbbl@single\endcname

```

```
9381 \chardef\bbl@bidimode\z@
9382 <</Emulate LaTeX>>

A proxy file:

9383 <{*plain}
9384 \input babel.def
9385 </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).