

Babel

Code

Version 24.1.40072
2024/02/04

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TeX

pdfTeX

LuaTeX

XeTeX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	Multiple languages	7
3.2	The Package File (\LaTeX , <code>babel.sty</code>)	8
3.3	<code>base</code>	9
3.4	<code>key=value</code> options and other general option	10
3.5	Conditional loading of shorthands	11
3.6	Interlude for Plain	13
4	Multiple languages	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	Hooks	25
4.4	Setting up language files	27
4.5	Shorthands	29
4.6	Language attributes	38
4.7	Support for saving macro definitions	39
4.8	Short tags	41
4.9	Hyphens	41
4.10	Multiencoding strings	43
4.11	Macros common to a number of languages	48
4.12	Making glyphs available	48
4.12.1	Quotation marks	48
4.12.2	Letters	50
4.12.3	Shorthands for quotation marks	50
4.12.4	Umlauts and tremas	51
4.13	Layout	52
4.14	Load engine specific macros	53
4.15	Creating and modifying languages	53
5	Adjusting the Babel behavior	76
5.1	Cross referencing macros	78
5.2	Marks	81
5.3	Preventing clashes with other packages	82
5.3.1	<code>ifthen</code>	82
5.3.2	<code>varioref</code>	83
5.3.3	<code>hhline</code>	83
5.4	Encoding and fonts	84
5.5	Basic bidi support	85
5.6	Local Language Configuration	88
5.7	Language options	89
6	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	92
7	Loading hyphenation patterns	96
8	Font handling with <code>fontspec</code>	100
9	Hooks for XeTeX and LuaTeX	103
9.1	XeTeX	103

10	Support for interchar	105
10.1	Layout	107
10.2	8-bit TeX	109
10.3	LuaTeX	109
10.4	Southeast Asian scripts	115
10.5	CJK line breaking	117
10.6	Arabic justification	119
10.7	Common stuff	123
10.8	Automatic fonts and ids switching	123
10.9	Bidi	129
10.10	Layout	131
10.11	Lua: transforms	139
10.12	Lua: Auto bidi with <code>basic</code> and <code>basic-r</code>	147
11	Data for CJK	158
12	The ‘nil’ language	159
13	Calendars	160
13.1	Islamic	160
13.2	Hebrew	161
13.3	Persian	165
13.4	Coptic and Ethiopic	166
13.5	Buddhist	167
14	Support for Plain TeX (<code>plain.def</code>)	168
14.1	Not renaming <code>hyphen.tex</code>	168
14.2	Emulating some L ^A T _E X features	169
14.3	General tools	169
14.4	Encoding related macros	173
15	Acknowledgements	176

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

Code documentation is still under revision.

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 (eg, 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 250 languages. They are distributed as a separate zip file, not packed as dtx. Most 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 (eg, 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=24.1.40072>>
2 <<date=2024/02/04>>
```

Do not use the following macros in ldf files. They may change in the future. This applies mainly to those recently added for replacing, trimming and looping. The older ones, like `\bbl@afterfi`, will not change.

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in babel.def and in babel.sty, which means in \LaTeX is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

`\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 `\ifcsname`, which is more efficient, and does not waste

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

memory. Defined inside a group, to avoid `\ifcsname` being implicitly set to `\relax` by the `\csname` test.

```

56 \beginingroup
57   \gdef\bbl@ifunset#1{%
58     \expandafter\ifx\csname#1\endcsname\relax
59       \expandafter\@firstoftwo
60     \else
61       \expandafter\@secondoftwo
62     \fi}
63 \bbl@ifunset{ifcsname}%
64 {}%
65 {\gdef\bbl@ifunset#1{%
66   \ifcsname#1\endcsname
67     \expandafter\ifx\csname#1\endcsname\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, ie, 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` (ie, 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}}

```

`\bbl@replace` Returns implicitly `\toks@` with the modified string.

```

101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}
103 \def\bbl@replace@aux##1#2##2#2{%

```

```

104 \ifx\bbl@nil##2%
105 \toks@{\expandafter{\the\toks@##1}%
106 \else
107 \toks@{\expandafter{\the\toks@##1#3}%
108 \bbl@afterfi
109 \bbl@replace@aux##2#2%
110 \fi}%
111 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112 \edef#1{\the\toks@}}

```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, 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).

```

113 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
114 \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
115 \def\bbl@tempa{#1}%
116 \def\bbl@tempb{#2}%
117 \def\bbl@tempe{#3}}
118 \def\bbl@sreplace#1#2#3{%
119 \begingroup
120 \expandafter\bbl@parsedef\meaning#1\relax
121 \def\bbl@tempc{#2}%
122 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
123 \def\bbl@tempd{#3}%
124 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
125 \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
126 \ifin@
127 \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
128 \def\bbl@tempc{% Expanded an executed below as 'uplevel'
129 \\makeatletter % "internal" macros with @ are assumed
130 \\scantokens{%
131 \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132 \catcode64=\the\catcode64\relax}% Restore @
133 \else
134 \let\bbl@tempc\empty % Not \relax
135 \fi
136 \bbl@exp{% For the 'uplevel' assignments
137 \endgroup
138 \bbl@tempc}} % empty or expand to set #1 with changes
139 \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.

```

140 \def\bbl@ifsamestring#1#2{%
141 \begingroup
142 \protected@edef\bbl@tempb{#1}%
143 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
144 \protected@edef\bbl@tempc{#2}%
145 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
146 \ifx\bbl@tempb\bbl@tempc
147 \aftergroup\@firstoftwo
148 \else
149 \aftergroup\@secondoftwo
150 \fi
151 \endgroup}
152 \chardef\bbl@engine=%
153 \ifx\directlua\undefined
154 \ifx\XeTeXinputencoding\undefined
155 \z@

```

```

156 \else
157 \tw@
158 \fi
159 \else
160 \@ne
161 \fi

```

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

```

162 \def\bbl@bsphack{%
163 \ifhmode
164 \hskip\z@skip
165 \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166 \else
167 \let\bbl@esphack\@empty
168 \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`.

```

169 \def\bbl@cased{%
170 \ifx\oe\OE
171 \expandafter\in@\expandafter
172 {\expandafter\OE\expandafter}\expandafter{\oe}%
173 \ifin@
174 \bbl@afterelse\expandafter\MakeUppercase
175 \else
176 \bbl@afterfi\expandafter\MakeLowercase
177 \fi
178 \else
179 \expandafter\@firstofone
180 \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`).

```

181 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
182 \toks@\expandafter\expandafter\expandafter{%
183 \csname extras\language\endcsname}%
184 \bbl@exp{\in@{#1}}{\the\toks@}}%
185 \ifin@\else
186 \@temptokena{#2}%
187 \edef\bbl@tempc{\the\@temptokena\the\toks@}%
188 \toks@\expandafter{\bbl@tempc#3}%
189 \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
190 \fi}
191 <</Basic macros>>

```

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

```

192 <<(*Make sure ProvidesFile is defined)>> \equiv
193 \ifx\ProvidesFile\@undefined
194 \def\ProvidesFile#1[#2 #3 #4]{%
195 \wlog{File: #1 #4 #3 <#2>}%
196 \let\ProvidesFile\@undefined}
197 \fi
198 <</Make sure ProvidesFile is defined>>

```

3.1 Multiple languages

`\language` Plain \TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in `switch.def` and `hyphen.cfg`; the latter may seem redundant, but remember `babel` doesn't require loading `switch.def` in the format.

```

199 <<(*Define core switching macros)>> \equiv

```



```

200 \ifx\language\@undefined
201   \csname newcount\endcsname\language
202 \fi
203 <</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.

```

204 <<*Define core switching macros>> ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 <</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 The Package File (\LaTeX , `babel.sty`)

```

208 (*package)
209 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
210 \ProvidesPackage{babel}[<<date>> v<<version>> The Babel package]

```

Start with some “private” debugging tool, and then define macros for errors.

```

211 \@ifpackagewith{babel}{debug}
212   {\providecommand\bbbl@trace[1]{\message{^^J[ #1 ]}}%
213    \let\bbbl@debug\@firstofone
214    \ifx\directlua\@undefined\else
215      \directlua{ Babel = Babel or {}
216                Babel.debug = true }%
217      \input{babel-debug.tex}%
218    \fi}
219 {\providecommand\bbbl@trace[1]{}%
220  \let\bbbl@debug\@gobble
221  \ifx\directlua\@undefined\else
222    \directlua{ Babel = Babel or {}
223              Babel.debug = false }%
224  \fi}
225 \def\bbbl@error#1{% Implicit #2#3#4
226   \begingroup
227     \catcode`\=0 \catcode`\==12 \catcode`\`=12
228     \input errbabel.def
229   \endgroup
230   \bbbl@error{#1}}
231 \def\bbbl@warning#1{%
232   \begingroup
233     \def\{\MessageBreak}%
234     \PackageWarning{babel}{#1}%
235   \endgroup}
236 \def\bbbl@infowarn#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \PackageNote{babel}{#1}%
240   \endgroup}
241 \def\bbbl@info#1{%
242   \begingroup
243     \def\{\MessageBreak}%
244     \PackageInfo{babel}{#1}%

```

```
245 \endgroup}
```

This file also takes care of a number of compatibility issues with other packages and defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

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.

```
246 <<Basic macros>>
247 \ifpackagewith{babel}{silent}
248 {\let\bb@info\@gobble
249 \let\bb@infowarn\@gobble
250 \let\bb@warning\@gobble}
251 {}
252 %
253 \def\AfterBabelLanguage#1{%
254 \global\expandafter\bb@add\csname#1.ldf-h@k\endcsname}%
```

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

```
255 \ifx\bb@languages\@undefined\else
256 \begingroup
257 \catcode\^^I=12
258 \ifpackagewith{babel}{showlanguages}{%
259 \begingroup
260 \def\bb@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
261 \wlog{<*languages>}%
262 \bb@languages
263 \wlog{</languages>}%
264 \endgroup}{%
265 \endgroup
266 \def\bb@elt#1#2#3#4{%
267 \ifnum#2=\z@
268 \gdef\bb@nulllanguage{#1}%
269 \def\bb@elt##1##2##3##4{%
270 \fi}%
271 \bb@languages
272 \fi%
```

3.3 base

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

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

```
273 \bb@trace{Defining option 'base'}
274 \ifpackagewith{babel}{base}{%
275 \let\bb@onlyswitch\@empty
276 \let\bb@provide@locale\relax
277 \input babel.def
278 \let\bb@onlyswitch\@undefined
279 \ifx\directlua\@undefined
280 \DeclareOption*{\bb@patterns{\CurrentOption}}%
281 \else
282 \input luababel.def
283 \DeclareOption*{\bb@patterns@lua{\CurrentOption}}%
284 \fi
285 \DeclareOption{base}{}%
286 \DeclareOption{showlanguages}{}%
287 \ProcessOptions
288 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
289 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
290 \global\let\@ifl@ter@\@ifl@ter
291 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@}%
```

```
292 \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`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\@for` or load `keyval`, for example.

```
293 \bbl@trace{key=value and another general options}
294 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
295 \def\bbl@tempb#1.#2{% Remove trailing dot
296   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
297 \def\bbl@tempe#1=#2\@{
298   \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}}
299 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
300   \ifx\@empty#2%
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302   \else
303     \in@{,provide=}{, #1}%
304     \ifin@
305       \edef\bbl@tempc{%
306         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
307     \else
308       \in@{ $modifiers$ }{ $#1$ }% TODO. Allow spaces.
309       \ifin@
310         \bbl@tempe#2\@
311       \else
312         \in@{=}{ #1}%
313         \ifin@
314           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
315         \else
316           \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
317           \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
318         \fi
319       \fi
320     \fi
321   \fi}
322 \let\bbl@tempc\@empty
323 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
324 \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.

```
325 \DeclareOption{KeepShorthandsActive}{}
326 \DeclareOption{activeacute}{}
327 \DeclareOption{activegrave}{}
328 \DeclareOption{debug}{}
329 \DeclareOption{noconfigs}{}
330 \DeclareOption{showlanguages}{}
331 \DeclareOption{silent}{}
332 % \DeclareOption{mono}{}
333 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
334 \chardef\bbl@iniflag\z@
335 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
336 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
337 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
338 % A separate option
339 \let\bbl@autoload@options\@empty
340 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
341 % Don't use. Experimental. TODO.
342 \newif\ifbbl@single
343 \DeclareOption{selectors=off}{\bbl@singletrue}
```

```
344 <<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 `<key>=<value>`, 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.

```
345 \let\bbl@opt@shorthands\@nnil
346 \let\bbl@opt@config\@nnil
347 \let\bbl@opt@main\@nnil
348 \let\bbl@opt@headfoot\@nnil
349 \let\bbl@opt@layout\@nnil
350 \let\bbl@opt@provide\@nnil
```

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

```
351 \def\bbl@tempa#1=#2\bbl@tempa{%
352   \bbl@csarg\ifx{opt#1}\@nnil
353     \bbl@csarg\edef{opt#1}{#2}%
354   \else
355     \bbl@error{bad-package-option}{#1}{#2}{}%
356   \fi}
```

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

```
357 \let\bbl@language@opts\@empty
358 \DeclareOption*{%
359   \bbl@xin@{\string=}{\CurrentOption}%
360   \ifin@
361     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
362   \else
363     \bbl@add@list\bbl@language@opts{\CurrentOption}%
364   \fi}
```

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

```
365 \ProcessOptions*
366 \ifx\bbl@opt@provide\@nnil
367   \let\bbl@opt@provide\@empty % %%% MOVE above
368 \else
369   \chardef\bbl@iniflag\@ne
370   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
371     \in@{,provide,}{, #1,}%
372     \ifin@
373       \def\bbl@opt@provide{#2}%
374       \bbl@replace\bbl@opt@provide{;}{,}%
375     \fi}
376 \fi
377 %
```

3.5 Conditional loading of shorthands

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

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

```
378 \bbl@trace{Conditional loading of shorthands}
379 \def\bbl@sh@string#1{%
380   \ifx#1\@empty\else
381     \ifx#1t\string~%
382     \else\ifx#1c\string,%
383     \else\string#1%
384   \fi\fi
385   \expandafter\bbl@sh@string
386 \fi}
```

```

387 \ifx\bbbl@opt@shorthands\@nnil
388   \def\bbbl@ifshorthand#1#2#3{#2}%
389 \else\ifx\bbbl@opt@shorthands\@empty
390   \def\bbbl@ifshorthand#1#2#3{#3}%
391 \else

```

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

```

392   \def\bbbl@ifshorthand#1{%
393     \bbbl@xin@\string#1}{\bbbl@opt@shorthands}%
394     \ifin@
395     \expandafter\@firstoftwo
396     \else
397     \expandafter\@secondoftwo
398     \fi}

```

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

```

399   \edef\bbbl@opt@shorthands{%
400     \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

401   \bbbl@ifshorthand{'}%
402     {\PassOptionsToPackage{activeacute}{babel}}{}
403   \bbbl@ifshorthand{`}%
404     {\PassOptionsToPackage{activegrave}{babel}}{}
405 \fi\fi

```

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

```

406 \ifx\bbbl@opt@headfoot\@nnil\else
407   \g@addto@macro\@resetactivechars{%
408     \set@typeset@protect
409     \expandafter\select@language\x\expandafter{\bbbl@opt@headfoot}%
410     \let\protect\noexpand}
411 \fi

```

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

```

412 \ifx\bbbl@opt@safe\@undefined
413   \def\bbbl@opt@safe{BR}
414   % \let\bbbl@opt@safe\@empty % Pending of \cite
415 \fi

```

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

```

416 \bbbl@trace{Defining IfBabelLayout}
417 \ifx\bbbl@opt@layout\@nnil
418   \newcommand\IfBabelLayout[3]{#3}%
419 \else
420   \bbbl@exp{\bbbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
421     \in{, layout, },{, #1, }%
422     \ifin@
423       \def\bbbl@opt@layout{#2}%
424       \bbbl@replace\bbbl@opt@layout{ }{.}%
425       \fi}
426   \newcommand\IfBabelLayout[1]{%
427     \@expandtwoargs\in{.#1.}{.\bbbl@opt@layout.}%
428     \ifin@
429     \expandafter\@firstoftwo
430     \else
431     \expandafter\@secondoftwo
432     \fi}
433 \fi
434 </package>
435 <*core>

```

3.6 Interlude for Plain

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*.

```
436 \ifx\ldf@quit\undefined\else
437 \endinput\fi % Same line!
438 <<Make sure ProvidesFile is defined>>
439 \ProvidesFile{babel.def}[\<date>] v\<version> Babel common definitions]
440 \ifx\AtBeginDocument\undefined % TODO. change test.
441 <<Emulate LaTeX>>
442 \fi
443 <<Basic macros>>
```

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

```
444 </core>
445 <*package | core>
```

4 Multiple languages

This is not a separate file (switch.def) anymore.

Plain \TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```
446 \def\bbbl@version{\<version>}
447 \def\bbbl@date{\<date>}
448 <<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.

```
449 \def\adddialect#1#2{%
450   \global\chardef#1#2\relax
451   \bbbl@usehooks{adddialect}{\#1}{\#2}}%
452   \begingroup
453     \count@#1\relax
454     \def\bbbl@elt##1##2###3###4{%
455       \ifnum\count@=##2\relax
456         \edef\bbbl@tempa{\expandafter\@gobbletwo\string#1}%
457         \bbbl@info{Hyphen rules for '\expandafter\@gobble\bbbl@tempa'
458                   set to \expandafter\string\csname l@##1\endcsname\\%
459                   (\string\language\the\count@). Reported}%
460         \def\bbbl@elt####1####2####3####4{%
461           \fi}%
462       \bbbl@cs{languages}%
463     \endgroup}
```

`\bbbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error. The argument of `\bbbl@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.

```
464 \def\bbbl@fixname#1{%
465   \begingroup
466     \def\bbbl@tempe{l@}%
467     \edef\bbbl@tempd{\noexpand\@ifundefined{\noexpand\bbbl@tempe#1}}%
468     \bbbl@tempd
469     {\lowercase\expandafter{\bbbl@tempd}}%
470     {\uppercase\expandafter{\bbbl@tempd}}%
471     \@empty
472     {\edef\bbbl@tempd{\def\noexpand#1{\#1}}%
473       \uppercase\expandafter{\bbbl@tempd}}}%

```

```

474      {\edef\bbl@tempd{\def\noexpand#1{#1}}%
475       \lowercase\expandafter{\bbl@tempd}}}%
476      \@empty
477      \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
478      \bbl@tempd
479      \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
480 \def\bbl@iflanguage#1{%
481   \ifundefined{l@#1}{\@nolannerr{#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 BCP47 code.

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

```

482 \def\bbl@bcpcase#1#2#3#4\@@#5{%
483   \ifx\@empty#3%
484     \uppercase{\def#5{#1#2}}%
485   \else
486     \uppercase{\def#5{#1}}%
487     \lowercase{\edef#5{#5#2#3#4}}%
488   \fi}
489 \def\bbl@bcpllookup#1-#2-#3-#4\@@{%
490   \let\bbl@bcp\relax
491   \lowercase{\def\bbl@tempa{#1}}%
492   \ifx\@empty#2%
493     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
494   \else\ifx\@empty#3%
495     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
496     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
497       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
498     {}%
499     \ifx\bbl@bcp\relax
500       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
501     \fi
502   \else
503     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
504     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
505     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
506       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
507     {}%
508     \ifx\bbl@bcp\relax
509       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
510         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
511       {}%
512     \fi
513     \ifx\bbl@bcp\relax
514       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
515         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
516       {}%
517     \fi
518     \ifx\bbl@bcp\relax
519       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
520     \fi
521   \fi\fi}
522 \let\bbl@initoload\relax
523 (-core)
524 \def\bbl@provide@locale{%
525   \ifx\babelprovide\@undefined
526     \bbl@error{base-on-the-fly}{}}}%
527   \fi
528   \let\bbl@auxname\language % Still necessary. TODO
529   \bbl@ifunset{\bbl@bcp@map@language}{}% Move uplevel??
530   {\edef\language{\@nameuse{\bbl@bcp@map@language}}}%

```

```

531 \ifbbl@bcpallowed
532   \expandafter\ifx\csname date\language\endcsname\relax
533     \expandafter
534     \bbl@bcplookup\language-\@empty-\@empty-\@empty\@@
535     \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
536       \edef\language{\bbl@bcp@prefix\bbl@bcp}%
537       \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
538       \expandafter\ifx\csname date\language\endcsname\relax
539         \let\bbl@initoload\bbl@bcp
540         \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
541         \let\bbl@initoload\relax
542       \fi
543       \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
544     \fi
545   \fi
546 \fi
547 \expandafter\ifx\csname date\language\endcsname\relax
548   \IfFileExists{babel-\language.tex}%
549   {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
550   {}%
551 \fi}
552 (+core)

```

\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.

```

553 \def\iflanguage#1{%
554   \bbl@iflanguage{#1}{%
555     \ifnum\csname l@#1\endcsname=\language
556       \expandafter\@firstoftwo
557     \else
558       \expandafter\@secondoftwo
559     \fi}}

```

4.1 Selecting the language

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

```

560 \let\bbl@select@type\z@
561 \edef\selectlanguage{%
562   \noexpand\protect
563   \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`.

```

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

```

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

```

565 \let\xstring\string

```

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

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

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

```
566 \def\bbl@language@stack{}
```

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

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

```
\bbl@pop@language
567 \def\bbl@push@language{%
568   \ifx\language\@undefined\else
569     \ifx\currentgrouplevel\@undefined
570       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
571     \else
572       \ifnum\currentgrouplevel=\z@
573         \xdef\bbl@language@stack{\language+}%
574       \else
575         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
576       \fi
577     \fi
578   \fi}
```

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

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

```
579 \def\bbl@pop@lang#1+#2\@{%
580   \edef\language{#1}%
581   \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).

```
582 \let\bbl@ifrestoring\@secondoftwo
583 \def\bbl@pop@language{%
584   \expandafter\bbl@pop@lang\bbl@language@stack\@
585   \let\bbl@ifrestoring\@firstoftwo
586   \expandafter\bbl@set@language\expandafter{\language}%
587   \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).

```
588 \chardef\localeid\z@
589 \def\bbl@id@last{0} % No real need for a new counter
590 \def\bbl@id@assign{%
591   \bbl@ifunset\bbl@id@\language}%
592   {\count@\bbl@id@last\relax
593     \advance\count@\@ne
594     \bbl@csarg\chardef{id@\language}\count@
595     \edef\bbl@id@last{\the\count@}%
596     \ifcase\bbl@engine\or
597       \directlua{
598         Babel = Babel or {}
599         Babel.locale_props = Babel.locale_props or {}
600         Babel.locale_props[\bbl@id@last] = {}
601         Babel.locale_props[\bbl@id@last].name = '\language'}
```

```

602     }%
603     \fi}%
604     }%
605     \chardef\localeid\bbl@cl{id@}}

```

The unprotected part of `\selectlanguage`.

```

606 \expandafter\def\csname selectlanguage \endcsname#1{%
607   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw\fi
608   \bbl@push@language
609   \aftergroup\bbl@pop@language
610   \bbl@set@language{#1}}

```

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

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

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

```

611 \def\BabelContentsFiles{toc,lof,lot}
612 \def\bbl@set@language#1{% from selectlanguage, pop@
613   % The old buggy way. Preserved for compatibility.
614   \edef\language#1%
615   \ifnum\escapechar=\expandafter`\string#1\@empty
616     \else\string#1\@empty\fi}%
617   \ifcat\relax\noexpand#1%
618     \expandafter\ifx\csname date\language\endcsname\relax
619       \edef\language#1%
620       \let\localename\language
621     \else
622       \bbl@info{Using '\string\language' instead of 'language' is\\%
623         deprecated. If what you want is to use a\\%
624         macro containing the actual locale, make\\%
625         sure it does not not match any language.\\%
626         Reported}%
627       \ifx\scantokens\@undefined
628         \def\localename{??}%
629       \else
630         \scantokens\expandafter{\expandafter
631           \def\expandafter\localename\expandafter{\language}}%
632       \fi
633     \fi
634   \else
635     \def\localename{#1}% This one has the correct catcodes
636   \fi
637   \select@language{\language}%
638   % write to auxs
639   \expandafter\ifx\csname date\language\endcsname\relax\else
640     \if@filesw
641       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
642         \bbl@savelastskip
643         \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
644         \bbl@restorelastskip
645       \fi
646       \bbl@usehooks{write}{}%
647     \fi
648   \fi}
649 %

```

```

650 \let\bbl@restorelastskip\relax
651 \let\bbl@savelastskip\relax
652 %
653 \newif\ifbbl@bcpallowed
654 \bbl@bcpallowedfalse
655 \def\select@language#1{% from set@, babel@aux
656   \ifx\bbl@selectorname\@empty
657     \def\bbl@selectorname{select}%
658   % set hymap
659   \fi
660   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
661   % set name
662   \edef\language#1}%
663   \bbl@fixname\language
664   % TODO. name@map must be here?
665   \bbl@provide@locale
666   \bbl@iflanguage\language{%
667     \let\bbl@select@type\z@
668     \expandafter\bbl@switch\expandafter{\language}}
669 \def\babel@aux#1#2{%
670   \select@language{#1}%
671   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
672     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
673 \def\babel@toc#1#2{%
674   \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<lang>` command at definition time by expanding the `\csname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

```

675 \newif\ifbbl@usedategroup
676 \let\bbl@savedextras\@empty
677 \def\bbl@switch#1{% from select@, foreign@
678   % make sure there is info for the language if so requested
679   \bbl@ensureinfo{#1}%
680   % restore
681   \originalTeX
682   \expandafter\def\expandafter\originalTeX\expandafter{%
683     \csname noextras#1\endcsname
684     \let\originalTeX\@empty
685     \babel@beginsave}%
686   \bbl@usehooks{afterreset}}}%
687   \languageshorthands{none}%
688   % set the locale id
689   \bbl@id@assign
690   % switch captions, date
691   \bbl@bsphack
692   \ifcase\bbl@select@type
693     \csname captions#1\endcsname\relax
694     \csname date#1\endcsname\relax
695   \else
696     \bbl@xin@{,captions,}{,\bbl@select@opts,}%
697   \ifin@

```

```

698     \csname captions#1\endcsname\relax
699     \fi
700     \bbl@xin@{,date,}{,\bbl@select@opts,}%
701     \ifin@ % if \foreign... within \<lang>date
702     \csname date#1\endcsname\relax
703     \fi
704     \fi
705     \bbl@esphack
706     % switch extras
707     \csname bbl@preextras@#1\endcsname
708     \bbl@usehooks{beforeextras}{}%
709     \csname extras#1\endcsname\relax
710     \bbl@usehooks{afterextras}{}%
711     % > babel-ensure
712     % > babel-sh-<short>
713     % > babel-bidi
714     % > babel-fontspec
715     \let\bbl@savedextras\@empty
716     % hyphenation - case mapping
717     \ifcase\bbl@opt@hyphenmap\or
718     \def\BabelLower##1##2{\lccode##1=##2\relax}%
719     \ifnum\bbl@hymap>4\else
720     \csname\language\name @bbl@hyphenmap\endcsname
721     \fi
722     \chardef\bbl@opt@hyphenmap\z@
723     \else
724     \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
725     \csname\language\name @bbl@hyphenmap\endcsname
726     \fi
727     \fi
728     \let\bbl@hymap\@ccclv
729     % hyphenation - select rules
730     \ifnum\csname l@\language\endcsname=\l@unhyphenated
731     \edef\bbl@tempa{u}%
732     \else
733     \edef\bbl@tempa{\bbl@cl{\lnbrk}}%
734     \fi
735     % linebreaking - handle u, e, k (v in the future)
736     \bbl@xin@{/u}{/\bbl@tempa}%
737     \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
738     \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
739     \ifin@ \else \bbl@xin@{/p}{/\bbl@tempa} \fi % padding (eg, Tibetan)
740     \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
741     \ifin@
742     % unhyphenated/kashida/elongated/padding = allow stretching
743     \language\l@unhyphenated
744     \babel@savevariable\emergencystretch
745     \emergencystretch\maxdimen
746     \babel@savevariable\hbadness
747     \hbadness\@M
748     \else
749     % other = select patterns
750     \bbl@patterns{#1}%
751     \fi
752     % hyphenation - mins
753     \babel@savevariable\lefthyphenmin
754     \babel@savevariable\righthyphenmin
755     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
756     \set@hyphenmins\tw@\thr@@\relax
757     \else
758     \expandafter\expandafter\expandafter\set@hyphenmins
759     \csname #1hyphenmins\endcsname\relax
760     \fi

```

```

761 % reset selector name
762 \let\bbl@selectorname\@empty}

```

`otherlanguage (env.)` The `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```

763 \long\def\otherlanguage#1{%
764   \def\bbl@selectorname{other}%
765   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
766   \csname selectlanguage \endcsname{#1}%
767   \ignorespaces}

```

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

```

768 \long\def\endotherlanguage{%
769   \global\@ignoretrue\ignorespaces}

```

`otherlanguage* (env.)` The `otherlanguage` environment 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’. This environment makes use of `\foreign@language`.

```

770 \expandafter\def\csname otherlanguage*\endcsname{%
771   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
772 \def\bbl@otherlanguage@s[#1]#2{%
773   \def\bbl@selectorname{other*}%
774   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
775   \def\bbl@select@opts{#1}%
776   \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”.

```

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

```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. 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{lang}` 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.

```

778 \providecommand\bbl@beforeforeign{}
779 \edef\foreignlanguage{%
780   \noexpand\protect
781   \expandafter\noexpand\csname foreignlanguage \endcsname}
782 \expandafter\def\csname foreignlanguage \endcsname{%
783   \@ifstar\bbl@foreign@s\bbl@foreign@x}
784 \providecommand\bbl@foreign@x[3][]{%

```

```

785 \begingroup
786 \def\bbl@selectorname{foreign}%
787 \def\bbl@select@opts{#1}%
788 \let\BabelText\@firstofone
789 \bbl@beforeforeign
790 \foreign@language{#2}%
791 \bbl@usehooks{foreign}{}%
792 \BabelText{#3}% Now in horizontal mode!
793 \endgroup}
794 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
795 \begingroup
796 {\par}%
797 \def\bbl@selectorname{foreign*}%
798 \let\bbl@select@opts\@empty
799 \let\BabelText\@firstofone
800 \foreign@language{#1}%
801 \bbl@usehooks{foreign*}{}%
802 \bbl@dirparastext
803 \BabelText{#2}% Still in vertical mode!
804 {\par}%
805 \endgroup}

```

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

```

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

```

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

```

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

```

854 \def\hyphenrules#1{%
855   \edef\bb@tempf{#1}%
856   \bb@fixname\bb@tempf
857   \bb@iflanguage\bb@tempf{%
858     \expandafter\bb@patterns\expandafter{\bb@tempf}%
859     \ifx\languageshorthands\@undefined\else
860       \languageshorthands{none}%
861     \fi
862     \expandafter\ifx\csname\bb@tempf hyphenmins\endcsname\relax
863       \set@hyphenmins\tw@\thr@@\relax
864     \else
865       \expandafter\expandafter\expandafter\set@hyphenmins
866       \csname\bb@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 `\langhyphenmins` 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 $\text{\LaTeX 2}_{\epsilon}$. When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by `babel`. Depending on the format, ie, on 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\csname 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` 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@warning{%
909 \@backslashchar#1 not set for '\language'. Please,\\%
910 define it after the language has been loaded\\%
911 (typically in the preamble) with:\\%
912 \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
913 Feel free to contribute on github.com/latex3/babel.\\%
914 Reported}}
915 \def\bbl@tentative{\protect\bbl@tentative@i}
916 \def\bbl@tentative@i#1{%
917 \bbl@warning{%
918 Some functions for '#1' are tentative.\\%

```



```

919   They might not work as expected and their behavior\\%
920   could change in the future.\\%
921   Reported}}
922 \def\nolanerr#1{\bbl@error{undefined-language}{#1}{}{}}
923 \def\nopatterns#1{%
924   \bbl@warning
925   {No hyphenation patterns were preloaded for\\%
926    the language '#1' into the format.\\%
927    Please, configure your TeX system to add them and\\%
928    rebuild the format. Now I will use the patterns\\%
929    preloaded for \bbl@nulllanguage\space instead}}
930 \let\bbl@usehooks@gobbletwo
931 \ifx\bbl@onlyswitch@empty\endinput\fi
932 % Here ended switch.def

```

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

```

933 \ifx\directlua@undefined\else
934   \ifx\bbl@luapatterns@undefined
935     \input luababel.def
936   \fi
937 \fi
938 \bbl@trace{Compatibility with language.def}
939 \ifx\bbl@languages@undefined
940   \ifx\directlua@undefined
941     \openin1 = language.def % TODO. Remove hardcoded number
942     \ifeof1
943       \closein1
944       \message{I couldn't find the file language.def}
945     \else
946       \closein1
947       \begingroup
948         \def\addlanguage#1#2#3#4#5{%
949           \expandafter\ifx\csname lang@#1\endcsname\relax\else
950             \global\expandafter\let\csname l@#1\expandafter\endcsname
951             \csname lang@#1\endcsname
952           \fi}%
953         \def\uselanguage#1{%
954           \input language.def
955         \endgroup
956       \fi
957     \fi
958     \chardef\l@english\z@
959 \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.

```

960 \def\addto#1#2{%
961   \ifx#1@undefined
962     \def#1{#2}%
963   \else
964     \ifx#1\relax
965       \def#1{#2}%
966     \else
967       {\toks@\expandafter{#1#2}%
968        \xdef#1{\the\toks@}}%
969     \fi
970   \fi}

```

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.

```

971 \def\bbl@withactive#1#2{%
972   \begingroup
973     \lccode`~=`#2\relax
974     \lowercase{\endgroup#1~}}

```

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

```

975 \def\bbl@redefine#1{%
976   \edef\bbl@tempa{\bbl@stripslash#1}%
977   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
978   \expandafter\def\csname\bbl@tempa\endcsname{
979   \@onlypreamble\bbl@redefine

```

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

```

980 \def\bbl@redefine@long#1{%
981   \edef\bbl@tempa{\bbl@stripslash#1}%
982   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
983   \long\expandafter\def\csname\bbl@tempa\endcsname{
984   \@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`.

```

985 \def\bbl@redefineroobust#1{%
986   \edef\bbl@tempa{\bbl@stripslash#1}%
987   \bbl@ifunset{\bbl@tempa\space}%
988     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
989      \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
990     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
991     \namedef{\bbl@tempa\space}}
992 \@onlypreamble\bbl@redefineroobust

```

4.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. `\bbl@usehooks` is the commands used by babel to execute hooks defined for an event.

```

993 \bbl@trace{Hooks}
994 \newcommand\AddBabelHook[3][]{%
995   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}}%
996   \def\bbl@tempa##1,##2,##3\empty{\def\bbl@tempb{##2}}%
997   \expandafter\bbl@tempa\bbl@evargs,##3=,\@empty
998   \bbl@ifunset{\bbl@ev@#2@#3@#1}%
999     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1000     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1001   \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1002 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1003 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1004 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1005 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1006   \ifx\UseHook\undefined\else\UseHook{babel/*/#2}\fi
1007   \def\bbl@elth##1{%
1008     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}}%
1009   \bbl@cs{ev@#2@#3}%
1010   \ifx\language\undefined\else % Test required for Plain (?)
1011     \ifx\UseHook\undefined\else\UseHook{babel/#1/#2}\fi
1012     \def\bbl@elth##1{%
1013       \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1@#3}}}%
1014     \bbl@cs{ev@#2@#1}%
1015   \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).

```

1016 \def\bbl@evargs{,% <- don't delete this comma
1017   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1018   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1019   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1020   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1021   beforestart=0,language=2,begindocument=1}
1022 \ifx\NewHook\undefined\else % Test for Plain (?)
1023   \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1024   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1025 \fi

```

`\babelensure` The user command just parses the optional argument and creates a new macro named `\bbl@e@{language}`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro `\bbl@e@{language}` contains `\bbl@ensure{(include)}{(exclude)}{(fontenc)}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the fontenc is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```

1026 \bbl@trace{Defining babelensure}
1027 \newcommand\babelensure[2][{}]{%
1028   \AddBabelHook{babel-ensure}{afterextras}{%
1029     \ifcase\bbl@select@type
1030       \bbl@cl{e}%
1031     \fi}%
1032   \begingroup
1033     \let\bbl@ens@include\empty
1034     \let\bbl@ens@exclude\empty
1035     \def\bbl@ens@fontenc{\relax}%
1036     \def\bbl@tempb##1{%
1037       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1038     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1039     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1040     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1041     \def\bbl@tempc{\bbl@ensure}%
1042     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1043       \expandafter{\bbl@ens@include}}%
1044     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1045       \expandafter{\bbl@ens@exclude}}%
1046     \toks@{\expandafter{\bbl@tempc}%
1047       \bbl@exp}%
1048   \endgroup
1049   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}%
1050 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1051   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1052     \ifx##1\undefined % 3.32 - Don't assume the macro exists
1053       \edef##1{\noexpand\bbl@nocaption
1054         {\bbl@stripslash##1}{\language\bbl@stripslash##1}}%
1055     \fi
1056     \ifx##1\@empty\else
1057       \in@{##1}{#2}%
1058     \fin@else
1059       \bbl@ifunset{\bbl@ensure@\language}%
1060       {\bbl@exp}%
1061       \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
1062         \\\foreignlanguage{\language}%
1063         {\ifx\relax#3\else
1064           \\\fontencoding{#3}\selectfont
1065         \fi

```

```

1066         #####1}}}%
1067     {}%
1068     \toks@\expandafter{##1}%
1069     \edef##1{%
1070         \bbl@csarg\noexpand{ensure@\language}%
1071         {\the\toks@}}%
1072     \fi
1073     \expandafter\bbl@tempb
1074     \fi}%
1075     \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1076     \def\bbl@tempa##1{% elt for include list
1077         \ifx##1\@empty\else
1078             \bbl@csarg\in@{ensure@\language\expandafter}\expandafter{##1}%
1079             \ifin@\else
1080                 \bbl@tempb##1\@empty
1081             \fi
1082             \expandafter\bbl@tempa
1083         \fi}%
1084     \bbl@tempa#1\@empty}
1085 \def\bbl@captionslist{%
1086     \prefacename\refname\abstractname\bibname\chaptername\appendixname
1087     \contentsname\listfigurename\listtablename\indexname\figurename
1088     \tablename\partname\enclname\ccname\headtoname\pagename\seename
1089     \alsoname\proofname\glossaryname}

```

4.4 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 `@`-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`.

```

1090 \bbl@trace{Macros for setting language files up}
1091 \def\bbl@ldfinit{%
1092     \let\bbl@screset\@empty
1093     \let\BabelStrings\bbl@opt@string
1094     \let\BabelOptions\@empty
1095     \let\BabelLanguages\relax
1096     \ifx\originalTeX\@undefined
1097         \let\originalTeX\@empty
1098     \else
1099         \originalTeX
1100     \fi}
1101 \def\LdfInit#1#2{%
1102     \chardef\atcatcode=\catcode\@
1103     \catcode\@=11\relax
1104     \chardef\eqcatcode=\catcode\=
1105     \catcode\==12\relax
1106     \expandafter\if\expandafter\@backslashchar
1107         \expandafter\@car\string#2\@nil

```

```

1108 \ifx#2\undefined\else
1109 \ldf@quit{#1}%
1110 \fi
1111 \else
1112 \expandafter\ifx\csname#2\endcsname\relax\else
1113 \ldf@quit{#1}%
1114 \fi
1115 \fi
1116 \bbl@ldfinit}

```

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

```

1117 \def\ldf@quit#1{%
1118 \expandafter\main@language\expandafter{#1}%
1119 \catcode\@=\atcatcode \let\atcatcode\relax
1120 \catcode\==\eqcatcode \let\eqcatcode\relax
1121 \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.

```

1122 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1123 \bbl@afterlang
1124 \let\bbl@afterlang\relax
1125 \let\BabelModifiers\relax
1126 \let\bbl@screset\relax}%
1127 \def\ldf@finish#1{%
1128 \loadlocalcfg{#1}%
1129 \bbl@afterldf{#1}%
1130 \expandafter\main@language\expandafter{#1}%
1131 \catcode\@=\atcatcode \let\atcatcode\relax
1132 \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 *LT_εX*.

```

1133 \@onlypreamble\LdfInit
1134 \@onlypreamble\ldf@quit
1135 \@onlypreamble\ldf@finish

```

`\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.

```

1136 \def\main@language#1{%
1137 \def\bbl@main@language{#1}%
1138 \let\language\name\bbl@main@language % TODO. Set localename
1139 \bbl@id@assign
1140 \bbl@patterns{\language\name}}

```

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

```

1141 \def\bbl@beforestart{%
1142 \def\@nolanerr##1{%
1143 \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1144 \bbl@usehooks{beforestart}}}%
1145 \global\let\bbl@beforestart\relax}
1146 \AtBeginDocument{%
1147 {\@nameuse{bbl@beforestart}}% Group!
1148 \if@filesw
1149 \providecommand\babel@aux[2]{}%
1150 \immediate\write\@mainaux{%
1151 \string\providecommand\string\babel@aux[2]{}%

```

```

1152 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1153 \fi
1154 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1155 <-core>
1156 \ifx\bbl@normalsf\@empty
1157 \ifnum\sfcodes\@.=\@m
1158 \let\normalsfcodes\frenchspacing
1159 \else
1160 \let\normalsfcodes\nonfrenchspacing
1161 \fi
1162 \else
1163 \let\normalsfcodes\bbl@normalsf
1164 \fi
1165 <+core>
1166 \ifbbl@single % must go after the line above.
1167 \renewcommand\selectlanguage[1]{}%
1168 \renewcommand\foreignlanguage[2]{#2}%
1169 \global\let\babel@aux\@gobbletwo % Also as flag
1170 \fi}
1171 <-core>
1172 \AddToHook{begindocument/before}{%
1173 \let\bbl@normalsf\normalsfcodes
1174 \let\normalsfcodes\relax} % Hack, to delay the setting
1175 <+core>
1176 \ifcase\bbl@engine\or
1177 \AtBeginDocument{\pagedir\bodydir} % TODO - a better place
1178 \fi

```

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

```

1179 \def\select@language@x#1{%
1180 \ifcase\bbl@select@type
1181 \bbl@ifsamestring\language@name{#1}{\select@language{#1}}%
1182 \else
1183 \select@language{#1}%
1184 \fi}

```

4.5 Shorthands

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if \TeX is used). It is used only at one place, namely when `\initiate@active@char` is called (which is ignored if the char has been made active before). Because `\@sanitize` can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with `\nfss@catcodes`, added in 3.10.

```

1185 \bbl@trace{Shorhands}
1186 \def\bbl@add@special#1{% 1:a macro like \, \?, etc.
1187 \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1188 \bbl@ifunset{@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1189 \ifx\nfss@catcodes\undefined\else % TODO - same for above
1190 \begingroup
1191 \catcode`#1\active
1192 \nfss@catcodes
1193 \ifnum\catcode`#1=\active
1194 \endgroup
1195 \bbl@add\nfss@catcodes{\@makeother#1}%
1196 \else
1197 \endgroup
1198 \fi
1199 \fi}

```

`\bbl@remove@special` The companion of the former macro is `\bbl@remove@special`. It removes a character from the set macros `\dospecials` and `\@sanitize`, but it is not used at all in the babel core.

```

1200 \def\bbl@remove@special#1{%
1201   \begingroup
1202     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1203       \else\noexpand##1\noexpand##2\fi}%
1204     \def\do{\x\do}%
1205     \def\@makeother{\x\@makeother}%
1206   \edef\x{\endgroup
1207     \def\noexpand\dospecials{\dospecials}%
1208     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1209       \def\noexpand\@sanitize{\@sanitize}%
1210     \fi}%
1211   \x}

```

`\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 "` (ie, with the original `"`); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\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).

```

1212 \def\bbl@active@def#1#2#3#4{%
1213   \@namedef{#3#1}{%
1214     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1215       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1216     \else
1217       \bbl@afterfi\csname#2@sh@#1\endcsname
1218     \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.

```

1219   \long\@namedef{#3@arg#1}##1{%
1220     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1221       \bbl@afterelse\csname#4#1\endcsname##1%
1222     \else
1223       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1224     \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.

```

1225 \def\initiate@active@char#1{%
1226   \bbl@ifunset{active@char\string#1}%
1227   {\bbl@withactive
1228     {\expandafter\initiate@active@char\expandafter}#1\string#1#1}%
1229   {}}

```

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).

```

1230 \def\@initiate@active@char#1#2#3{%
1231   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1232   \ifx#1\@undefined

```

```

1233 \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1234 \else
1235 \bbl@csarg\let{oridef@#2}#1%
1236 \bbl@csarg\edef{oridef@#2}{%
1237 \let\noexpand#1%
1238 \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1239 \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*").

```

1240 \ifx#1#3\relax
1241 \expandafter\let\csname normal@char#2\endcsname#3%
1242 \else
1243 \bbl@info{Making #2 an active character}%
1244 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1245 \@namedef{normal@char#2}{%
1246 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1247 \else
1248 \@namedef{normal@char#2}{#3}%
1249 \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).

```

1250 \bbl@restoreactive{#2}%
1251 \AtBeginDocument{%
1252 \catcode`#2\active
1253 \if@filesw
1254 \immediate\write\@mainaux{\catcode`\string#2\active}%
1255 \fi}%
1256 \expandafter\bbl@add@special\csname#2\endcsname
1257 \catcode`#2\active
1258 \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⟩`).

```

1259 \let\bbl@tempa\@firstoftwo
1260 \if\string^#2%
1261 \def\bbl@tempa{\noexpand\textormath}%
1262 \else
1263 \ifx\bbl@mathnormal\@undefined\else
1264 \let\bbl@tempa\bbl@mathnormal
1265 \fi
1266 \fi
1267 \expandafter\edef\csname active@char#2\endcsname{%
1268 \bbl@tempa
1269 {\noexpand\if@safe@actives
1270 \noexpand\expandafter
1271 \expandafter\noexpand\csname normal@char#2\endcsname
1272 \noexpand\else
1273 \noexpand\expandafter
1274 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1275 \noexpand\fi}%
1276 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1277 \bbl@csarg\edef{doactive#2}{%

```



```
1278 \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

```
\active@prefix <char> \normal@char<char>
```

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

```
1279 \bbl@csarg\edef{active@#2}{%
1280 \noexpand\active@prefix\noexpand#1%
1281 \expandafter\noexpand\csname active@char#2\endcsname}%
1282 \bbl@csarg\edef{normal@#2}{%
1283 \noexpand\active@prefix\noexpand#1%
1284 \expandafter\noexpand\csname normal@char#2\endcsname}%
1285 \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.

```
1286 \bbl@active@def#2\user@group{user@active}{language@active}%
1287 \bbl@active@def#2\language@group{language@active}{system@active}%
1288 \bbl@active@def#2\system@group{system@active}{normal@char}%
```

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

```
1289 \expandafter\edef\csname\user@group @sh#2@\endcsname
1290 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1291 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1292 {\expandafter\noexpand\csname user@active#2\endcsname}%
```

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

```
1293 \if\string'#2%
1294 \let\prim@s\bbl@prim@s
1295 \let\active@math@prime#1%
1296 \fi
1297 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1298 <<More package options>> ≡
1299 \DeclareOption{math=active}{}
1300 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1301 <</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*.

```
1302 \@ifpackagewith{babel}{KeepShorthandsActive}%
1303 {\let\bbl@restoreactive@gobble}%
1304 {\def\bbl@restoreactive#1{%
1305 \bbl@exp{%
1306 \\\AfterBabelLanguage\\\CurrentOption
1307 {\catcode`#1=\the\catcode`#1\relax}%
1308 \\\AtEndOfPackage
1309 {\catcode`#1=\the\catcode`#1\relax}}}%
1310 \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.

```

1311 \def\bbl@sh@select#1#2{%
1312   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1313     \bbl@afterelse\bbl@scndcs
1314   \else
1315     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1316   \fi}

```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protect`s 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.

```

1317 \begingroup
1318 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1319 {\gdef\active@prefix#1{%
1320   \ifx\protect\@typeset@protect
1321   \else
1322     \ifx\protect\@unexpandable@protect
1323       \noexpand#1%
1324     \else
1325       \protect#1%
1326     \fi
1327   \expandafter\@gobble
1328   \fi}}
1329 {\gdef\active@prefix#1{%
1330   \ifincsname
1331     \string#1%
1332     \expandafter\@gobble
1333   \else
1334     \ifx\protect\@typeset@protect
1335     \else
1336       \ifx\protect\@unexpandable@protect
1337         \noexpand#1%
1338       \else
1339         \protect#1%
1340       \fi
1341     \expandafter\expandafter\expandafter\@gobble
1342     \fi
1343   \fi}}
1344 \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@activestru`), 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`).

```

1345 \newif\if@safe@actives
1346 \@safe@activefalse

```

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

```

1347 \def\bbl@restore@actives{\if@safe@actives\@safe@activefalse\fi}

```

`\bbl@activate` 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`.

```

1348 \chardef\bbl@activated\z@
1349 \def\bbl@activate#1{%
1350   \chardef\bbl@activated\@ne
1351   \bbl@withactive{\expandafter\let\expandafter}#1%
1352     \csname bbl@active@\string#1\endcsname}
1353 \def\bbl@deactivate#1{%
1354   \chardef\bbl@activated\tw@
1355   \bbl@withactive{\expandafter\let\expandafter}#1%
1356     \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1357 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1358 \def\bbl@scndcs#1#2{\csname#2\endcsname}

```

`\declare@shorthand` The command `\declare@shorthand` is 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.

```

1359 \def\babel@texpdf#1#2#3#4{%
1360   \ifx\texorpdfstring\@undefined
1361     \textormath{#1}{#3}%
1362   \else
1363     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1364     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1365   \fi}
1366 %
1367 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1368 \def\@decl@short#1#2#3\@nil#4{%
1369   \def\bbl@tempa{#3}%
1370   \ifx\bbl@tempa\@empty
1371     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1372     \bbl@ifunset{#1@sh@\string#2@}{}%
1373     {\def\bbl@tempa{#4}%
1374       \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1375       \else
1376         \bbl@info
1377           {Redefining #1 shorthand \string#2\\%
1378            in language \CurrentOption}%
1379       \fi}%
1380     \@namedef{#1@sh@\string#2@}{#4}%
1381   \else
1382     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1383     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1384     {\def\bbl@tempa{#4}%
1385       \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1386       \else
1387         \bbl@info
1388           {Redefining #1 shorthand \string#2\string#3\\%
1389            in language \CurrentOption}%
1390       \fi}%
1391     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1392   \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.

```
1393 \def\textormath{%
1394   \ifmmode
1395     \expandafter\@secondoftwo
1396   \else
1397     \expandafter\@firstoftwo
1398   \fi}
```

`\user@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 `\language@group` group ‘english’ and have a system group called ‘system’.

```
1399 \def\user@group{user}
1400 \def\language@group{english} % TODO. I don't like defaults
1401 \def\system@group{system}
```

`\useshorthands` This is the user level macro. It initializes and activates the character for use as a shorthand character (ie, 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.

```
1402 \def\useshorthands{%
1403   \ifstar\bb@usesh@s{\bb@usesh@x{}}
1404 \def\bb@usesh@s#1{%
1405   \bb@usesh@x
1406     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1407     {#1}}
1408 \def\bb@usesh@x#1#2{%
1409   \bb@ifshorthand{#2}%
1410     {\def\user@group{user}%
1411       \initiate@active@char{#2}%
1412       #1%
1413       \bb@activate{#2}}%
1414     {\bb@error{shorthand-is-off}{#2}{}}}
```

`\defineshorthand` Currently we only support two groups of user level shorthands, named internally `user` and `user@<lang>` (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.

```
1415 \def\user@language@group{user@language@group}
1416 \def\bb@set@user@generic#1#2{%
1417   \bb@ifunset{user@generic@active#1}%
1418     {\bb@active@def#1\user@language@group{user@active}{user@generic@active}%
1419       \bb@active@def#1\user@group{user@generic@active}{language@active}%
1420       \expandafter\edef\csname#2@sh@#1@\endcsname{%
1421         \expandafter\noexpand\csname normal@char#1\endcsname}%
1422       \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1423         \expandafter\noexpand\csname user@active#1\endcsname}}%
1424   \@empty}
1425 \newcommand\defineshorthand[3][user]{%
1426   \edef\bb@tempa{\zap@space#1 \@empty}%
1427   \bb@for\bb@tempb\bb@tempa{%
1428     \if*\expandafter\@car\bb@tempb\@nil
1429       \edef\bb@tempb{user@\expandafter\@gobble\bb@tempb}%
1430       \@expandtwoargs
1431       \bb@set@user@generic{\expandafter\string\@car#2\@nil}\bb@tempb
1432     \fi
1433     \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. [TODO].

```
1434 \def\languageshorthands#1{\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`.

```

1435 \def\aliasshorthand#1#2{%
1436   \bbl@ifshorthand{#2}%
1437   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1438     \ifx\document\@notprerr
1439       \@notshorthand{#2}%
1440     \else
1441       \initiate@active@char{#2}%
1442       \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1443       \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1444       \bbl@activate{#2}%
1445     \fi
1446   \fi}%
1447   {\bbl@error{shorthand-is-off}{#2}{}}}
```

`\@notshorthand`

```

1448 \def\@notshorthand#1{\bbl@error{not-a-shorthand}{#1}{}}
```

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

```

1449 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1450 \DeclareRobustCommand*\shorthandoff{%
1451   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1452 \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.

```

1453 \def\bbl@switch@sh#1#2{%
1454   \ifx#2\@nnil\else
1455     \bbl@ifunset{\bbl@active@\string#2}%
1456     {\bbl@error{not-a-shorthand-b}{#2}{}}%
1457     {\ifcase#1%   off, on, off*
1458       \catcode`#2\relax
1459     \or
1460       \catcode`#2\active
1461       \bbl@ifunset{\bbl@shdef@\string#2}%
1462       {}%
1463       {\bbl@withactive{\expandafter\let\expandafter}#2%
1464         \csname bbl@shdef@\string#2\endcsname
1465         \bbl@csarg\let{shdef@\string#2}\relax}%
1466       \ifcase\bbl@activated\or
1467         \bbl@activate{#2}%
1468       \else
1469         \bbl@deactivate{#2}%
1470       \fi
1471     \or
1472       \bbl@ifunset{\bbl@shdef@\string#2}%
1473       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2%
1474       {}%
1475       \csname bbl@oricat@\string#2\endcsname
1476       \csname bbl@oridef@\string#2\endcsname
1477       \fi}%
1478     \bbl@afterfi\bbl@switch@sh#1%
1479   \fi}
```

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

```

1480 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1481 \def\bbl@putsh#1{%
1482   \bbl@ifunset{\bbl@active@\string#1}%
1483   {\bbl@putsh@i#1@empty\@nnil}%
1484   {\csname bbl@active@\string#1\endcsname}}
1485 \def\bbl@putsh@i#1#2\@nnil{%
1486   \csname\language@group @sh@\string#1@%
1487   \ifx\@empty#2\else\string#2@\fi\endcsname}
1488 %
1489 \ifx\bbl@opt@shorthands\@nnil\else
1490   \let\bbl@s@initiate@active@char\initiate@active@char
1491   \def\initiate@active@char#1{%
1492     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1493   \let\bbl@s@switch@sh\bbl@switch@sh
1494   \def\bbl@switch@sh#1#2{%
1495     \ifx#2\@nnil\else
1496       \bbl@afterfi
1497       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1498       \fi}
1499   \let\bbl@s@activate\bbl@activate
1500   \def\bbl@activate#1{%
1501     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1502   \let\bbl@s@deactivate\bbl@deactivate
1503   \def\bbl@deactivate#1{%
1504     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1505   \fi

```

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

```

1506 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

`\bbl@prim@s` One of the internal macros that are involved in substituting `\prime` for each right quote in
`\bbl@pr@m@s` 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.

```

1507 \def\bbl@prim@s{%
1508   \prime\futurelet\@let@token\bbl@pr@m@s}
1509 \def\bbl@if@primes#1#2{%
1510   \ifx#1\@let@token
1511     \expandafter\@firstoftwo
1512   \else\ifx#2\@let@token
1513     \bbl@afterelse\expandafter\@firstoftwo
1514   \else
1515     \bbl@afterfi\expandafter\@secondoftwo
1516   \fi\fi}
1517 \begingroup
1518 \catcode`\^=7 \catcode`\*=\active \lccode`\*=`^
1519 \catcode`\'=12 \catcode`\"=\active \lccode`\"=`'
1520 \lowercase{%
1521   \gdef\bbl@pr@m@s{%
1522     \bbl@if@primes" '%
1523     \pr@@s
1524     {\bbl@if@primes*^ \pr@@t\egroup}}}
1525 \endgroup

```

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

```

1526 \initiate@active@char{~}
1527 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1528 \bbl@activate{~}

```

\OT1dqpos 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.

```

1529 \expandafter\def\csname OT1dqpos\endcsname{127}
1530 \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

```

1531 \ifx\f@encoding\@undefined
1532   \def\f@encoding{OT1}
1533 \fi

```

4.6 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.

```

1534 \bbl@trace{Language attributes}
1535 \newcommand\languageattribute[2]{%
1536   \def\bbl@tempc{#1}%
1537   \bbl@fixname\bbl@tempc
1538   \bbl@iflanguage\bbl@tempc{%
1539     \bbl@vforeach{#2}{%

```

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

```

1540     \ifx\bbl@known@attrs\@undefined
1541       \in@false
1542     \else
1543       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1544     \fi
1545     \ifin@
1546       \bbl@warning{%
1547         You have more than once selected the attribute '##1'\%
1548         for language #1. Reported}%
1549     \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.

```

1550       \bbl@exp{%
1551         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1552       \edef\bbl@tempa{\bbl@tempc-##1}%
1553       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
1554       {\csname\bbl@tempc @attr##1\endcsname}%
1555       {\@attrerr{\bbl@tempc}{##1}}%
1556     \fi}}%
1557 \@onlypreamble\languageattribute

```

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

```

1558 \newcommand*{\@attrerr}[2]{%
1559   \bbl@error{unknown-attribute}{#1}{#2}{}}

```

\bbl@declare@tribute 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}.

```

1560 \def\bbl@declare@ttribute#1#2#3{%
1561   \bbl@xin@{,#2,},{,\BabelModifiers,}%
1562   \ifin@
1563     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1564   \fi
1565   \bbl@add@list\bbl@attributes{#1-#2}%
1566   \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.

```

1567 \def\bbl@ifattributeset#1#2#3#4{%
1568   \ifx\bbl@known@attrs\@undefined
1569     \in@false
1570   \else
1571     \bbl@xin@{,#1-#2,},{,\bbl@known@attrs,}%
1572   \fi
1573   \ifin@
1574     \bbl@afterelse#3%
1575   \else
1576     \bbl@afterfi#4%
1577   \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.

```

1578 \def\bbl@ifknown@ttrib#1#2{%
1579   \let\bbl@tempa\@secondoftwo
1580   \bbl@loopx\bbl@tempb{#2}{%
1581     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1582     \ifin@
1583       \let\bbl@tempa\@firstoftwo
1584     \else
1585       \fi}%
1586   \bbl@tempa}

```

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

```

1587 \def\bbl@clear@ttribs{%
1588   \ifx\bbl@attributes\@undefined\else
1589     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1590       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1591     \let\bbl@attributes\@undefined
1592   \fi}
1593 \def\bbl@clear@ttrib#1-#2.{%
1594   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1595 \AtBeginDocument{\bbl@clear@ttribs}

```

4.7 Support for saving macro definitions

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` The initialization of a new save cycle: reset the counter to zero.

`\babel@beginsave` 1596 `\bbl@trace{Macros for saving definitions}`
1597 `\def\babel@beginsave{\babel@savecnt\z@}`

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

1598 `\newcount\babel@savecnt`

1599 `\babel@beginsave`

`\babel@save` The macro `\babel@save{csname}` saves the current meaning of the control sequence `<csname>` to `\originalTeX`². 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.

```
1600 \def\babel@save#1{%
1601   \def\bbl@tempa{,{#1,}}% Clumsy, for Plain
1602   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1603     \expandafter{\expandafter,\bbl@savextras,}}%
1604   \expandafter\in@\bbl@tempa
1605   \ifin@ \else
1606     \bbl@add\bbl@savextras{,{#1,}}%
1607     \bbl@carg\let{babel@\number\babel@savecnt}#1\relax
1608     \toks@\expandafter{\originalTeX\let#1=}
1609     \bbl@exp{%
1610       \def\\originalTeX{\the\toks@<babel@\number\babel@savecnt>\relax}}%
1611     \advance\babel@savecnt@ne
1612   \fi}
1613 \def\babel@savevariable#1{%
1614   \toks@\expandafter{\originalTeX #1=}
1615   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}
```

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

```
1616 \def\bbl@frenchspacing{%
1617   \ifnum\the\sfcode`\.=\@m
1618     \let\bbl@nonfrenchspacing\relax
1619   \else
1620     \frenchspacing
1621     \let\bbl@nonfrenchspacing\nonfrenchspacing
1622   \fi}
1623 \let\bbl@nonfrenchspacing\nonfrenchspacing
1624 \let\bbl@elt\relax
1625 \edef\bbl@fs@chars{%
1626   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1627   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1628   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1629 \def\bbl@pre@fs{%
1630   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##2\relax}%
1631   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1632 \def\bbl@post@fs{%
1633   \bbl@save@sfcodes
1634   \edef\bbl@tempa{\bbl@cl{frspc}}%
1635   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1636   \if u\bbl@tempa % do nothing
1637   \else\if n\bbl@tempa % non french
1638     \def\bbl@elt##1##2##3{%
1639       \ifnum\sfcode`##1=##2\relax
1640       \babel@savevariable{\sfcode`##1}%

```

²`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```

1641     \sfcode`##1=##3\relax
1642   \fi}%
1643   \bbl@fs@chars
1644   \else\if y\bbl@tempa      % french
1645     \def\bbl@elt##1##2##3{%
1646       \ifnum\sfcode`##1=##3\relax
1647         \babel@savevariable{\sfcode`##1}%
1648         \sfcode`##1=##2\relax
1649       \fi}%
1650   \bbl@fs@chars
1651   \fi\fi\fi}

```

4.8 Short tags

`\babeltags` This macro is straightforward. After zapping spaces, we loop over the list and define the macros `\text<tag>` and `\<tag>`. Definitions are first expanded so that they don't contain `\csname` but the actual macro.

```

1652 \bbl@trace{Short tags}
1653 \def\babeltags#1{%
1654   \edef\bbl@tempa{\zap@space#1 \@empty}%
1655   \def\bbl@tempb##1=##2\@{ }%
1656   \edef\bbl@tempc{%
1657     \noexpand\newcommand
1658     \expandafter\noexpand\csname ##1\endcsname{%
1659       \noexpand\protect
1660       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1661     \noexpand\newcommand
1662     \expandafter\noexpand\csname text##1\endcsname{%
1663       \noexpand\foreignlanguage{##2}}
1664   \bbl@tempc}%
1665   \bbl@for\bbl@tempa\bbl@tempa{%
1666     \expandafter\bbl@tempb\bbl@tempa\@{ }

```

4.9 Hyphens

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

```

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

```

```

1691          #2}}}%
1692      \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`³.

```

1693 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1694 \def\bbl@t@one{Tl}
1695 \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`.

```

1696 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1697 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1698 \def\bbl@hyphen{%
1699   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1700 \def\bbl@hyphen@i#1#2{%
1701   \bbl@ifunset{\bbl@hy@#1#2\@empty}%
1702   {\csname bbl@#1usehyphen\endcsname\discretionary{#2}{#{#2}}}%
1703   {\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.

```

1704 \def\bbl@usehyphen#1{%
1705   \leavevmode
1706   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1707   \nobreak\hskip\z@skip}
1708 \def\bbl@usehyphen#1{%
1709   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1710 \def\bbl@hyphenchar{%
1711   \ifnum\hyphenchar\font=\m@ne
1712     \babelnullhyphen
1713   \else
1714     \char\hyphenchar\font
1715   \fi}

```

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

```

1716 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1717 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1718 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1719 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1720 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1721 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1722 \def\bbl@hy@repeat{%
1723   \bbl@usehyphen{%
1724     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1725 \def\bbl@hy@@repeat{%
1726   \bbl@usehyphen{%
1727     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1728 \def\bbl@hy@empty{\hskip\z@skip}
1729 \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.

```

1730 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#{#1}}\bbl@allowhyphens}

```

³`TEX` begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

4.10 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.

```
1731 \bbl@trace{Multiencoding strings}
1732 \def\bbl@tglobal#1{\global\let#1#1}
```

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

```
1733 <<{*More package options}>> ≡
1734 \DeclareOption{nocase}{}
1735 <</More package options>>
```

The following package options control the behavior of \SetString.

```
1736 <<{*More package options}>> ≡
1737 \let\bbl@opt@strings\@nnil % accept strings=value
1738 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1739 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1740 \def\BabelStringsDefault{generic}
1741 <</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.

```
1742 \@onlypreamble\StartBabelCommands
1743 \def\StartBabelCommands{%
1744   \begingroup
1745   \@tempcnta="7F
1746   \def\bbl@tempa{%
1747     \ifnum\@tempcnta>"FF\else
1748       \catcode\@tempcnta=11
1749       \advance\@tempcnta\@ne
1750       \expandafter\bbl@tempa
1751     \fi}%
1752   \bbl@tempa
1753   <<Macros local to BabelCommands>>
1754   \def\bbl@provstring##1##2{%
1755     \providecommand##1{##2}%
1756     \bbl@tglobal##1}%
1757   \global\let\bbl@scafter\@empty
1758   \let\StartBabelCommands\bbl@startcmds
1759   \ifx\BabelLanguages\relax
1760     \let\BabelLanguages\CurrentOption
1761   \fi
1762   \begingroup
1763   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1764   \StartBabelCommands}
1765 \def\bbl@startcmds{%
1766   \ifx\bbl@screset\@nnil\else
1767     \bbl@usehooks{stopcommands}{}%
1768   \fi
1769   \endgroup
1770   \begingroup
1771   \@ifstar
1772     {\ifx\bbl@opt@strings\@nnil
1773       \let\bbl@opt@strings\BabelStringsDefault
1774     \fi
1775     \bbl@startcmds@i}%
1776   \bbl@startcmds@i}
1777 \def\bbl@startcmds@i#1#2{%
1778   \edef\bbl@L{\zap@space#1 \@empty}}%
```

```

1779 \edef\bbbl@G{\zap@space#2 \@empty}%
1780 \bbbl@startcmds@ii}
1781 \let\bbbl@startcommands\StartBabelCommands

```

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

Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, 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 (ie, 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.

```

1782 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
1783 \let\SetString\@gobbletwo
1784 \let\bbbl@stringdef\@gobbletwo
1785 \let\AfterBabelCommands\@gobble
1786 \ifx\@empty#1%
1787 \def\bbbl@sc@label{generic}%
1788 \def\bbbl@encstring##1##2{%
1789 \ProvideTextCommandDefault##1{##2}%
1790 \bbbl@toglobal##1%
1791 \expandafter\bbbl@toglobal\csname\string? \string##1\endcsname}%
1792 \let\bbbl@sctest\in@true
1793 \else
1794 \let\bbbl@sc@charset\space % <- zapped below
1795 \let\bbbl@sc@fontenc\space % <- " "
1796 \def\bbbl@tempa##1=##2\@nil{%
1797 \bbbl@csarg\edef{sc@ \zap@space##1 \@empty}{##2 }}%
1798 \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
1799 \def\bbbl@tempa##1 ##2{% space -> comma
1800 ##1%
1801 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
1802 \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
1803 \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
1804 \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
1805 \def\bbbl@encstring##1##2{%
1806 \bbbl@foreach\bbbl@sc@fontenc{%
1807 \bbbl@ifunset{T@####1}%
1808 {}%
1809 {\ProvideTextCommand##1{####1}{##2}%
1810 \bbbl@toglobal##1%
1811 \expandafter
1812 \bbbl@toglobal\csname####1\string##1\endcsname}}}%
1813 \def\bbbl@sctest{%
1814 \bbbl@xin@{\bbbl@opt@strings,}{\bbbl@sc@label,\bbbl@sc@fontenc,}}%
1815 \fi
1816 \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
1817 \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded
1818 \let\AfterBabelCommands\bbbl@aftercmds
1819 \let\SetString\bbbl@setstring
1820 \let\bbbl@stringdef\bbbl@encstring
1821 \else % ie, strings=value
1822 \bbbl@sctest
1823 \ifin@
1824 \let\AfterBabelCommands\bbbl@aftercmds
1825 \let\SetString\bbbl@setstring
1826 \let\bbbl@stringdef\bbbl@provstring
1827 \fi\fi\fi
1828 \bbbl@scswitch
1829 \ifx\bbbl@G\@empty
1830 \def\SetString##1##2{%
1831 \bbbl@error{missing-group}{##1}{}}}%

```

```

1832 \fi
1833 \ifx\@empty#1%
1834 \bbl@usehooks{defaultcommands}{}%
1835 \else
1836 \@expandtwoargs
1837 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1838 \fi}

```

There are two versions of `\bbl@scswitch`. The first version is used when `ldfs` are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing. The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has been loaded).

```

1839 \def\bbl@forlang#1#2{%
1840 \bbl@for#1\bbl@L{%
1841 \bbl@xin@{,#1,}{,\BabelLanguages,}%
1842 \ifin@#2\relax\fi}}
1843 \def\bbl@scswitch{%
1844 \bbl@forlang\bbl@tempa{%
1845 \ifx\bbl@G\@empty\else
1846 \ifx\SetString\gobbletwo\else
1847 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1848 \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1849 \ifin@\else
1850 \global\expandafter\let\csname\bbl@GL\endcsname\undefined
1851 \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1852 \fi
1853 \fi
1854 \fi}}
1855 \AtEndOfPackage{%
1856 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1857 \let\bbl@scswitch\relax}
1858 \onlypreamble\EndBabelCommands
1859 \def\EndBabelCommands{%
1860 \bbl@usehooks{stopcommands}{}%
1861 \endgroup
1862 \endgroup
1863 \bbl@scafter}
1864 \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 (ie, 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.

```

1865 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1866 \bbl@forlang\bbl@tempa{%
1867 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1868 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1869 {\bbl@exp{%
1870 \global\\bbl@add\<\bbl@G\bbl@tempa>{\bbl@scset\\#1\<\bbl@LC>}}}%
1871 }%
1872 \def\BabelString{#2}%
1873 \bbl@usehooks{stringprocess}{}%
1874 \expandafter\bbl@stringdef
1875 \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}

```

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

```
1876 \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.

```
1877 <<*Macros local to BabelCommands>> ≡
1878 \def\SetStringLoop##1##2{%
1879   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1880   \count@\z@
1881   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1882     \advance\count@\@ne
1883     \toks@\expandafter{\bbl@tempa}%
1884     \bbl@exp{%
1885       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1886       \count@=\the\count@\relax}}}%
1887 <</Macros local to BabelCommands>>
```

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

```
1888 \def\bbl@aftercmds#1{%
1889   \toks@\expandafter{\bbl@scafter#1}%
1890   \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.

```
1891 <<*Macros local to BabelCommands>> ≡
1892 \newcommand\SetCase[3][]{%
1893   \def\bbl@tempa####1####2{%
1894     \ifx####1\@empty\else
1895       \bbl@carg\bbl@add{extras\CurrentOption}{%
1896         \bbl@carg\babel@save{c__text_uppercase\_string####1_tl}%
1897         \bbl@carg\def{c__text_uppercase\_string####1_tl}{####2}%
1898         \bbl@carg\babel@save{c__text_lowercase\_string####2_tl}%
1899         \bbl@carg\def{c__text_lowercase\_string####2_tl}{####1}}%
1900       \expandafter\bbl@tempa
1901       \fi}%
1902   \bbl@tempa##1\@empty\@empty
1903   \bbl@carg\bbl@toglobal{extras\CurrentOption}}%
1904 <</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.

```
1905 <<*Macros local to BabelCommands>> ≡
1906 \newcommand\SetHyphenMap[1]{%
1907   \bbl@forlang\bbl@tempa{%
1908     \expandafter\bbl@stringdef
1909     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1910 <</Macros local to BabelCommands>>
```

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

```
1911 \newcommand\BabelLower[2]{% one to one.
1912   \ifnum\lccode#1=#2\else
1913     \babel@savevariable{\lccode#1}%
1914     \lccode#1=#2\relax
1915   \fi}
1916 \newcommand\BabelLowerMM[4]{% many-to-many
1917   \@tempcnta=#1\relax
1918   \@tempcntb=#4\relax
1919   \def\bbl@tempa{%
1920     \ifnum\@tempcnta>#2\else
1921       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1922       \advance\@tempcnta#3\relax
```

```

1923 \advance\@tempcntb#3\relax
1924 \expandafter\bb\@tempa
1925 \fi}%
1926 \bb\@tempa}
1927 \newcommand\BabelLowerM0[4]{% many-to-one
1928 \@tempcnta=#1\relax
1929 \def\bb\@tempa{%
1930 \ifnum\@tempcnta>#2\else
1931 \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1932 \advance\@tempcnta#3
1933 \expandafter\bb\@tempa
1934 \fi}%
1935 \bb\@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1936 <<{*More package options}> \equiv
1937 \DeclareOption{hyphenmap=off}{\chardef\bb\@opt@hyphenmap\z@}
1938 \DeclareOption{hyphenmap=first}{\chardef\bb\@opt@hyphenmap\@ne}
1939 \DeclareOption{hyphenmap=select}{\chardef\bb\@opt@hyphenmap\tw@}
1940 \DeclareOption{hyphenmap=other}{\chardef\bb\@opt@hyphenmap\thr@@}
1941 \DeclareOption{hyphenmap=other*}{\chardef\bb\@opt@hyphenmap4\relax}
1942 <</More package options>>

```

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

```

1943 \AtEndOfPackage{%
1944 \ifx\bb\@opt@hyphenmap\undefined
1945 \bb\@xin@{,}{\bb\@language@opts}%
1946 \chardef\bb\@opt@hyphenmap\ifin@4\else\@ne\fi
1947 \fi}

```

This sections ends with 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.

```

1948 \newcommand\setlocalecaption{% TODO. Catch typos.
1949 \@ifstar\bb\@setcaption@{\bb\@setcaption@x}
1950 \def\bb\@setcaption@x#1#2#3{% language caption-name string
1951 \bb\@trim@def\bb\@tempa{#2}%
1952 \bb\@xin@{.template}{\bb\@tempa}%
1953 \ifin@
1954 \bb\@ini@captions@template{#3}{#1}%
1955 \else
1956 \edef\bb\@tempd{%
1957 \expandafter\expandafter\expandafter
1958 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1959 \bb\@xin@
1960 {\expandafter\string\csname #2name\endcsname}%
1961 {\bb\@tempd}%
1962 \ifin@ % Renew caption
1963 \bb\@xin@{\string\bb\@scset}{\bb\@tempd}%
1964 \ifin@
1965 \bb\@exp{%
1966 \\\bb\@ifsamestring{\bb\@tempa}{\language\name}%
1967 {\\\bb\@scset\<#2name>\<#1#2name>}%
1968 {}}%
1969 \else % Old way converts to new way
1970 \bb\@ifunset{#1#2name}%
1971 {\bb\@exp{%
1972 \\\bb\@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
1973 \\\bb\@ifsamestring{\bb\@tempa}{\language\name}%
1974 {\def\<#2name>{\<#1#2name>}}}%
1975 {}}}%
1976 {}%
1977 \fi
1978 \else

```



```

1979 \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
1980 \ifin@ % New way
1981 \bbl@exp{%
1982 \\\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}}%
1983 \\\bbl@ifsamestring{\bbl@tempa}{\language\name}%
1984 {\bbl@scset\<#2name>\<#1#2name>}}%
1985 {}}%
1986 \else % Old way, but defined in the new way
1987 \bbl@exp{%
1988 \\\bbl@add\<captions#1>{\def\<#2name>\<#1#2name>}}%
1989 \\\bbl@ifsamestring{\bbl@tempa}{\language\name}%
1990 {\def\<#2name>\<#1#2name>}}%
1991 {}}%
1992 \fi%
1993 \fi
1994 \@namedef{#1#2name}{#3}%
1995 \toks@{\expandafter{\bbl@captionslist}}%
1996 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
1997 \ifin@ \else
1998 \bbl@exp{\bbl@add\<captionslist>\<#2name>}}%
1999 \bbl@toглобal\bbl@captionslist
2000 \fi
2001 \fi}
2002 % \def\bbl@setcaption@#1#2#3{ % TODO. Not yet implemented (w/o 'name')

```

4.11 Macros common to a number of languages

`\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.

```

2003 \bbl@trace{Macros related to glyphs}
2004 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2005 \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2006 \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.

```

2007 \def\save@sf@q#1{\leavevmode
2008 \begingroup
2009 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2010 \endgroup}

```

4.12 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`.

4.12.1 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.

```

2011 \ProvideTextCommand{\quotedblbase}{OT1}{%
2012 \save@sf@q{\set@low@box{\textquotedblright\}}%
2013 \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.

```

2014 \ProvideTextCommandDefault{\quotedblbase}{%
2015 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2016 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2017 \save@sf@q{\set@low@box{\textquoteright\}}%
2018 \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.

```
2019 \ProvideTextCommandDefault{\quotesinglbase}{%
2020   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2021 \ProvideTextCommand{\guillemetleft}{OT1}{%
2022   \ifmmode
2023     \ll
2024   \else
2025     \save@sf@q{\nobreak
2026       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2027   \fi}
2028 \ProvideTextCommand{\guillemetright}{OT1}{%
2029   \ifmmode
2030     \gg
2031   \else
2032     \save@sf@q{\nobreak
2033       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2034   \fi}
2035 \ProvideTextCommand{\guillemotleft}{OT1}{%
2036   \ifmmode
2037     \ll
2038   \else
2039     \save@sf@q{\nobreak
2040       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2041   \fi}
2042 \ProvideTextCommand{\guillemotright}{OT1}{%
2043   \ifmmode
2044     \gg
2045   \else
2046     \save@sf@q{\nobreak
2047       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2048   \fi}
```

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

```
2049 \ProvideTextCommandDefault{\guillemetleft}{%
2050   \UseTextSymbol{OT1}{\guillemetleft}}
2051 \ProvideTextCommandDefault{\guillemetright}{%
2052   \UseTextSymbol{OT1}{\guillemetright}}
2053 \ProvideTextCommandDefault{\guillemotleft}{%
2054   \UseTextSymbol{OT1}{\guillemotleft}}
2055 \ProvideTextCommandDefault{\guillemotright}{%
2056   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
\guilsinglright 2057 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2058   \ifmmode
2059     <%
2060   \else
2061     \save@sf@q{\nobreak
2062       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2063   \fi}
2064 \ProvideTextCommand{\guilsinglright}{OT1}{%
2065   \ifmmode
2066     >%
2067   \else
2068     \save@sf@q{\nobreak
2069       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2070   \fi}
```

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

```
2071 \ProvideTextCommandDefault{\guilsinglleft}{%
```

```

2072 \UseTextSymbol{OT1}{\guilsinglleft}}
2073 \ProvideTextCommandDefault{\guilsinglright}{%
2074 \UseTextSymbol{OT1}{\guilsinglright}}

```

4.12.2 Letters

\ij The dutch language uses the letter ‘ij’. It is available in T1 encoded fonts, but not in the OT1 encoded \IJ fonts. Therefore we fake it for the OT1 encoding.

```

2075 \DeclareTextCommand{\ij}{OT1}{%
2076 i\kern-0.02em\bb1@allowhyphens j}
2077 \DeclareTextCommand{\IJ}{OT1}{%
2078 I\kern-0.02em\bb1@allowhyphens J}
2079 \DeclareTextCommand{\ij}{T1}{\char188}
2080 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2081 \ProvideTextCommandDefault{\ij}{%
2082 \UseTextSymbol{OT1}{\ij}}
2083 \ProvideTextCommandDefault{\IJ}{%
2084 \UseTextSymbol{OT1}{\IJ}}

```

\dj The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in \DJ 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).

```

2085 \def\crrtic@{\hrule height0.1ex width0.3em}
2086 \def\crttic@{\hrule height0.1ex width0.33em}
2087 \def\ddj@{%
2088 \setbox0\hbox{d}\dimen@=\ht0
2089 \advance\dimen@lex
2090 \dimen@.45\dimen@
2091 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2092 \advance\dimen@ii.5ex
2093 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2094 \def\DDJ@{%
2095 \setbox0\hbox{D}\dimen@=.55\ht0
2096 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2097 \advance\dimen@ii.15ex % correction for the dash position
2098 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2099 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2100 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2101 %
2102 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2103 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2104 \ProvideTextCommandDefault{\dj}{%
2105 \UseTextSymbol{OT1}{\dj}}
2106 \ProvideTextCommandDefault{\DJ}{%
2107 \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.

```

2108 \DeclareTextCommand{\SS}{OT1}{SS}
2109 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

4.12.3 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` The ‘german’ single quotes.

```
\grq 2110 \ProvideTextCommandDefault{\glq}{%
      2111 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2112 \ProvideTextCommand{\grq}{T1}{%
      2113 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}%
2114 \ProvideTextCommand{\grq}{TU}{%
      2115 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2116 \ProvideTextCommand{\grq}{OT1}{%
      2117 \save@sf@q{\kern-.0125em
      2118 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
      2119 \kern.07em\relax}}
2120 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2121 \ProvideTextCommandDefault{\glqq}{%
      2122 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2123 \ProvideTextCommand{\grqq}{T1}{%
      2124 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2125 \ProvideTextCommand{\grqq}{TU}{%
      2126 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2127 \ProvideTextCommand{\grqq}{OT1}{%
      2128 \save@sf@q{\kern-.07em
      2129 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
      2130 \kern.07em\relax}}
2131 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2132 \ProvideTextCommandDefault{\flq}{%
      2133 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}%
2134 \ProvideTextCommandDefault{\frq}{%
      2135 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2136 \ProvideTextCommandDefault{\flqq}{%
      2137 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2138 \ProvideTextCommandDefault{\frqq}{%
      2139 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

4.12.4 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` To be able to provide both positions of `\` we provide two commands to switch the positioning, the
`\umlautlow` default will be `\umlauthigh` (the normal positioning).

```
2140 \def\umlauthigh{%
      2141 \def\bbl@umlauta##1{\leavevmode\bgroup%
      2142 \accent\csname\fontencoding dqpos\endcsname
      2143 ##1\bbl@allowhyphens\egroup}%
      2144 \let\bbl@umlaute\bbl@umlauta}
2145 \def\umlautlow{%
      2146 \def\bbl@umlauta{\protect\lower@umlaut}}
2147 \def\umlautelow{%
      2148 \def\bbl@umlaute{\protect\lower@umlaut}}
2149 \umlauthigh
```

`\lower@umlaut` The command `\lower@umlaut` is 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.

```
2150 \expandafter\ifx\csname U@D\endcsname\relax
2151   \csname newdimen\endcsname\U@D
2152 \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.

```
2153 \def\lower@umlaut#1{%
2154   \leavevmode\bgroup
2155     \U@D 1ex%
2156     {\setbox\z@\hbox{%
2157       \char\csname\fontencoding dqpos\endcsname}%
2158       \dimen@ -.45ex\advance\dimen@\ht\z@
2159       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2160     \accent\csname\fontencoding dqpos\endcsname
2161     \fontdimen5\font\U@D #1%
2162   \egroup}
```

For all vowels we declare `\` to be a composite command which uses `\bbl@umlauta` or `\bbl@umlaute` to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package `fontenc` with option `OT1` is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but `babel` sets them for *all* languages – you may want to redefine `\bbl@umlauta` and/or `\bbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```
2163 \AtBeginDocument{%
2164   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2165   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2166   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{i}}%
2167   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{i}}%
2168   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2169   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2170   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2171   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2172   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2173   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2174   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}}
```

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

```
2175 \ifx\l@english\undefined
2176   \chardef\l@english\z@
2177 \fi
2178 % The following is used to cancel rules in ini files (see Amharic).
2179 \ifx\l@unhyphenated\undefined
2180   \newlanguage\l@unhyphenated
2181 \fi
```

4.13 Layout

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

```
2182 \bbl@trace{Bidi layout}
2183 \providecommand\IfBabelLayout[3]{#3}%
2184 <-core>
2185 \newcommand\BabelPatchSection[1]{%
2186   \@ifundefined{#1}{}{%
```

```

2187 \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2188 \@namedef{#1}{%
2189 \ifstar{\bbl@presec{s{#1}}}%
2190 {\@dblarg{\bbl@presec{x{#1}}}}}
2191 \def\bbl@presec@x#1[#2]#3{%
2192 \bbl@exp{%
2193 \\\select@language@x{\bbl@main@language}%
2194 \\\bbl@cs{sspre@#1}%
2195 \\\bbl@cs{ss@#1}%
2196 [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2197 {\\\foreignlanguage{\language}{\unexpanded{#3}}}%
2198 \\\select@language@x{\language}}}
2199 \def\bbl@presec@s#1#2{%
2200 \bbl@exp{%
2201 \\\select@language@x{\bbl@main@language}%
2202 \\\bbl@cs{sspre@#1}%
2203 \\\bbl@cs{ss@#1}*%
2204 {\\\foreignlanguage{\language}{\unexpanded{#2}}}%
2205 \\\select@language@x{\language}}}
2206 \IfBabelLayout{sectioning}%
2207 {\BabelPatchSection{part}%
2208 \BabelPatchSection{chapter}%
2209 \BabelPatchSection{section}%
2210 \BabelPatchSection{subsection}%
2211 \BabelPatchSection{subsubsection}%
2212 \BabelPatchSection{paragraph}%
2213 \BabelPatchSection{subparagraph}%
2214 \def\babel@toc#1{%
2215 \select@language@x{\bbl@main@language}}}%
2216 \IfBabelLayout{captions}%
2217 {\BabelPatchSection{caption}}}%
2218 \<+core>

```

4.14 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.

```

2219 \bbl@trace{Input engine specific macros}
2220 \ifcase\bbl@engine
2221 \input txtbabel.def
2222 \or
2223 \input luababel.def
2224 \or
2225 \input xebabel.def
2226 \fi
2227 \providecommand\babelfont{\bbl@error{only-lua-xe}}{}{}{}
2228 \providecommand\babelprehyphenation{\bbl@error{only-lua}}{}{}{}
2229 \ifx\babelposthyphenation\undefined
2230 \let\babelposthyphenation\babelprehyphenation
2231 \let\babelpatterns\babelprehyphenation
2232 \let\babelcharproperty\babelprehyphenation
2233 \fi

```

4.15 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.

```

2234 \</package | core>
2235 \<*package>
2236 \bbl@trace{Creating languages and reading ini files}

```

```

2237 \let\bbl@extend@ini@gobble
2238 \newcommand\babelprovide[2][{}]{%
2239   \let\bbl@savelangname\language
2240   \edef\bbl@savelocaleid{\the\localeid}%
2241   % Set name and locale id
2242   \edef\language{#2}%
2243   \bbl@id@assign
2244   % Initialize keys
2245   \bbl@vforeach{captions,date,import,main,script,language,%
2246     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2247     mapdigits,intraspaces,intrapenalty,onchar,transforms,alpha,%
2248     Alph,labels,labels*,calendar,date,casing,interchar}%
2249     {\bbl@csarg\let{KVP@##1}\@nnil}%
2250   \global\let\bbl@release@transforms\@empty
2251   \global\let\bbl@release@casing\@empty
2252   \let\bbl@calendars\@empty
2253   \global\let\bbl@inidata\@empty
2254   \global\let\bbl@extend@ini@gobble
2255   \global\let\bbl@included@inis\@empty
2256   \gdef\bbl@key@list{;}%
2257   \bbl@forkv{#1}{%
2258     \in@{/}{##1}% With /, (re)sets a value in the ini
2259     \ifin@
2260       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2261       \bbl@renewinikey##1\@{##2}%
2262     \else
2263       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2264         \bbl@error{unknown-provide-key}{##1}{}%
2265       \fi
2266       \bbl@csarg\def{KVP@##1}{##2}%
2267     \fi}%
2268   \chardef\bbl@howloaded=0:none;1:ldf without ini;2:ini
2269   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel@#2}\@ne\tw@}%
2270   % == init ==
2271   \ifx\bbl@screset\@undefined
2272     \bbl@ldfinit
2273   \fi
2274   % == date (as option) ==
2275   % \ifx\bbl@KVP@date\@nnil\else
2276   % \fi
2277   % ==
2278   \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2279   \ifcase\bbl@howloaded
2280     \let\bbl@lbkflag\@empty % new
2281   \else
2282     \ifx\bbl@KVP@hyphenrules\@nnil\else
2283       \let\bbl@lbkflag\@empty
2284     \fi
2285     \ifx\bbl@KVP@import\@nnil\else
2286       \let\bbl@lbkflag\@empty
2287     \fi
2288   \fi
2289   % == import, captions ==
2290   \ifx\bbl@KVP@import\@nnil\else
2291     \bbl@exp{\bbl@ifblank{\bbl@KVP@import}}%
2292     {\ifx\bbl@initload\relax
2293       \begingroup
2294         \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2295         \bbl@input@texini{#2}%
2296       \endgroup
2297     \else
2298       \xdef\bbl@KVP@import{\bbl@initload}%
2299     \fi}%

```

```

2300     {}%
2301     \let\bbl@KVP@date\@empty
2302 \fi
2303 \let\bbl@KVP@captions@\bbl@KVP@captions % TODO. A dirty hack
2304 \ifx\bbl@KVP@captions\@nnil
2305     \let\bbl@KVP@captions\bbl@KVP@import
2306 \fi
2307 % ==
2308 \ifx\bbl@KVP@transforms\@nnil\else
2309     \bbl@replace\bbl@KVP@transforms{ }{,}%
2310 \fi
2311 % == Load ini ==
2312 \ifcase\bbl@howloaded
2313     \bbl@provide@new{#2}%
2314 \else
2315     \bbl@ifblank{#1}%
2316     {}% With \bbl@load@basic below
2317     {\bbl@provide@renew{#2}}%
2318 \fi
2319 % == include == TODO
2320 % \ifx\bbl@included@inis\@empty\else
2321 %     \bbl@replace\bbl@included@inis{ }{,}%
2322 %     \bbl@foreach\bbl@included@inis{%
2323 %         \openin\bbl@readstream=babel-##1.ini
2324 %         \bbl@extend@ini{#2}%
2325 %         \closein\bbl@readstream
2326 %     } \fi
2327 % Post tasks
2328 % -----
2329 % == subsequent calls after the first provide for a locale ==
2330 \ifx\bbl@inidata\@empty\else
2331     \bbl@extend@ini{#2}%
2332 \fi
2333 % == ensure captions ==
2334 \ifx\bbl@KVP@captions\@nnil\else
2335     \bbl@ifunset{\bbl@extracaps@#2}%
2336     {\bbl@exp{\bbl@babelensure[exclude=\\today]{#2}}}%
2337     {\bbl@exp{\bbl@babelensure[exclude=\\today,
2338         include=\[bbl@extracaps@#2]]{#2}}}%
2339     \bbl@ifunset{\bbl@ensure@\language}%
2340     {\bbl@exp{%
2341         \\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2342             \\foreignlanguage{\language}%
2343             {###1}}}%
2344     }%
2345     \bbl@exp{%
2346         \\bbl@tglobal\<bbl@ensure@\language>%
2347         \\bbl@tglobal\<bbl@ensure@\language\space>}%
2348 \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.

```

2349 \bbl@load@basic{#2}%
2350 % == script, language ==
2351 % Override the values from ini or defines them
2352 \ifx\bbl@KVP@script\@nnil\else
2353     \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2354 \fi
2355 \ifx\bbl@KVP@language\@nnil\else
2356     \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2357 \fi
2358 \ifcase\bbl@engine\or

```



```

2359 \bbl@ifunset{\bbl@chrng@{language}}{}%
2360 {\directlua{
2361   Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2362 \fi
2363 % == onchar ==
2364 \ifx\bbl@KVP@onchar\@nnil\else
2365   \bbl@luahyphenate
2366   \bbl@exp{%
2367     \AddToHook{env/document/before}{\select@language{#2}}}%
2368   \directlua{
2369     if Babel.locale_mapped == nil then
2370       Babel.locale_mapped = true
2371       Babel.linebreaking.add_before(Babel.locale_map, 1)
2372       Babel.loc_to_scr = {}
2373       Babel.chr_to_loc = Babel.chr_to_loc or {}
2374     end
2375     Babel.locale_props[\the\localeid].letters = false
2376   }%
2377   \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2378   \ifin@
2379     \directlua{
2380       Babel.locale_props[\the\localeid].letters = true
2381     }%
2382   \fi
2383   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2384   \ifin@
2385     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2386       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2387     \fi
2388     \bbl@exp{\bbl@add\bbl@starthyphens
2389       {\bbl@patterns@lua{language}}}%
2390     % TODO - error/warning if no script
2391     \directlua{
2392       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2393         Babel.loc_to_scr[\the\localeid] = Babel.script_blocks['\bbl@cl{sbcpr}']
2394         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@language}\space
2395       end
2396     }%
2397   \fi
2398   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2399   \ifin@
2400     \bbl@ifunset{\bbl@lsys@{language}}{\bbl@provide@lsys{language}}{}%
2401     \bbl@ifunset{\bbl@wdir@{language}}{\bbl@provide@dirs{language}}{}%
2402     \directlua{
2403       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2404         Babel.loc_to_scr[\the\localeid] =
2405           Babel.script_blocks['\bbl@cl{sbcpr}']
2406       end}%
2407     \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
2408       \AtBeginDocument{%
2409         \bbl@patchfont{\bbl@mapselect}%
2410         {\selectfont}}%
2411       \def\bbl@mapselect{%
2412         \let\bbl@mapselect\relax
2413         \edef\bbl@prefontid{\fontid\font}}%
2414       \def\bbl@mapdir##1{%
2415         {\def\language{##1}}%
2416         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2417         \bbl@switchfont
2418         \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2419           \directlua{
2420             Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2421               [\the\fontid\font\space]}%

```

```

2422         \fi}}%
2423     \fi
2424     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2425     \fi
2426     % TODO - catch non-valid values
2427 \fi
2428 % == mapfont ==
2429 % For bidi texts, to switch the font based on direction
2430 \ifx\bbl@KVP@mapfont\@nnil\else
2431     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}%
2432     {\bbl@error{unknown-mapfont}}{}{}%
2433     \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys@language}}%
2434     \bbl@ifunset{\bbl@wdir@language}{\bbl@provide@dirs@language}}%
2435     \ifx\bbl@mapselect\undefined % TODO. See onchar.
2436         \AtBeginDocument{%
2437             \bbl@patchfont{\bbl@mapselect}%
2438             {\selectfont}}%
2439         \def\bbl@mapselect{%
2440             \let\bbl@mapselect\relax
2441             \edef\bbl@prefontid{\fontid\font}%
2442         \def\bbl@mapdir##1{%
2443             {\def\language{##1}%
2444             \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2445             \bbl@switchfont
2446             \directlua{Babel.fontmap
2447                 [\the\csname bbl@wdir@##1\endcsname]%
2448                 [\bbl@prefontid]=\fontid\font}}}%
2449         \fi
2450     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2451 \fi
2452 % == Line breaking: intraspace, intrapenalty ==
2453 % For CJK, East Asian, Southeast Asian, if interspace in ini
2454 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2455     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2456 \fi
2457 \bbl@provide@intraspace
2458 % == Line breaking: CJK quotes == TODO -> @extras
2459 \ifcase\bbl@engine\or
2460     \bbl@xin@{/c}{\bbl@cl{\lnbrk}}%
2461     \ifin@
2462         \bbl@ifunset{\bbl@quote@language}}%
2463         {\directlua{
2464             Babel.locale_props[\the\localeid].cjk_quotes = {}
2465             local cs = 'op'
2466             for c in string.utfvalues(
2467                 [[\csname bbl@quote@language\endcsname]]) do
2468                 if Babel.cjk_characters[c].c == 'qu' then
2469                     Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2470                 end
2471                 cs = (cs == 'op') and 'cl' or 'op'
2472             end
2473         }}%
2474     \fi
2475 \fi
2476 % == Line breaking: justification ==
2477 \ifx\bbl@KVP@justification\@nnil\else
2478     \let\bbl@KVP@linebreaking\bbl@KVP@justification
2479 \fi
2480 \ifx\bbl@KVP@linebreaking\@nnil\else
2481     \bbl@xin@{\bbl@KVP@linebreaking,%
2482         {,elongated,kashida,cjk,padding,unhyphenated},%
2483         \ifin@
2484             \bbl@csarg\xdef

```

```

2485     {\lnbrk@{\language\name}{\expandafter\@car\bbk@KVP@linebreaking\@nnil}%
2486     \fi
2487     \fi
2488     \bbk@xin@{/e}{/\bbk@cl{\lnbrk}}}%
2489     \ifin@else\bbk@xin@{/k}{/\bbk@cl{\lnbrk}}\fi
2490     \ifin@\bbk@arabicjust\fi
2491     \bbk@xin@{/p}{/\bbk@cl{\lnbrk}}}%
2492     \ifin@\AtBeginDocument{\@nameuse{\bbk@tibetanjust}}\fi
2493     % == Line breaking: hyphenate.other.(locale|script) ==
2494     \ifx\bbk@lbfkflag\empty
2495         \bbk@ifunset{\bbk@hyotl@{\language\name}}{%
2496             {\bbk@csarg\bbk@replace{\hyotl@{\language\name}}{ }{,}%
2497             \bbk@startcommands*{\language\name}}{%
2498                 \bbk@csarg\bbk@foreach{\hyotl@{\language\name}}{%
2499                     \ifcase\bbk@engine
2500                     \ifnum##1<257
2501                         \SetHyphenMap{\BabelLower{##1}{##1}}%
2502                     \fi
2503                     \else
2504                         \SetHyphenMap{\BabelLower{##1}{##1}}%
2505                     \fi}%
2506                 \bbk@endcommands}%
2507             \bbk@ifunset{\bbk@hyots@{\language\name}}{%
2508                 {\bbk@csarg\bbk@replace{\hyots@{\language\name}}{ }{,}%
2509                 \bbk@csarg\bbk@foreach{\hyots@{\language\name}}{%
2510                     \ifcase\bbk@engine
2511                     \ifnum##1<257
2512                         \global\lccode##1=##1\relax
2513                     \fi
2514                     \else
2515                         \global\lccode##1=##1\relax
2516                     \fi}}%
2517             \fi
2518             % == Counters: maparabic ==
2519             % Native digits, if provided in ini (TeX level, xe and lua)
2520             \ifcase\bbk@engine\else
2521                 \bbk@ifunset{\bbk@dgnat@{\language\name}}{%
2522                     {\expandafter\ifx\csname\bbk@dgnat@{\language\name}\endcsname\empty\else
2523                     \expandafter\expandafter\expandafter
2524                     \bbk@setdigits\csname\bbk@dgnat@{\language\name}\endcsname
2525                     \ifx\bbk@KVP@maparabic\@nnil\else
2526                     \ifx\bbk@latinarabic\@undefined
2527                     \expandafter\let\expandafter\@arabic
2528                     \csname\bbk@counter@{\language\name}\endcsname
2529                     \else % ie, if layout=counters, which redefines \@arabic
2530                     \expandafter\let\expandafter\bbk@latinarabic
2531                     \csname\bbk@counter@{\language\name}\endcsname
2532                     \fi
2533                     \fi
2534                 \fi}%
2535             \fi
2536             % == Counters: mapdigits ==
2537             % > luababel.def
2538             % == Counters: alph, Alph ==
2539             \ifx\bbk@KVP@alph\@nnil\else
2540                 \bbk@exp{%
2541                     \\bbk@add\<\bbk@preextras@{\language\name}>%
2542                     \\babel@save\\@alph
2543                     \let\\@alph\<\bbk@cntr@\bbk@KVP@alph @{\language\name}>}}%
2544             \fi
2545             \ifx\bbk@KVP@Alph\@nnil\else
2546                 \bbk@exp{%
2547                     \\bbk@add\<\bbk@preextras@{\language\name}>%

```

```

2548      \\babel@save\\@Alph
2549      \let\\@Alph<bbl@cntr@bbl@KVP@Alph @\languagename>}}%
2550 \fi
2551 % == Casing ==
2552 \bbl@release@casing
2553 \ifx\bbl@KVP@casing\@nnil\else
2554   \bbl@csarg\xdef{casing@languagename}%
2555   {\@nameuse{bbl@casing@languagename}\bbl@maybextx\bbl@KVP@casing}%
2556 \fi
2557 % == Calendars ==
2558 \ifx\bbl@KVP@calendar\@nnil
2559   \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2560 \fi
2561 \def\bbl@tempe##1 ##2\@{ % Get first calendar
2562   \def\bbl@tempa{##1}}%
2563   \bbl@exp{\\bbl@tempe\bbl@KVP@calendar\space\\@}%
2564 \def\bbl@tempe##1.##2.##3\@{ %
2565   \def\bbl@tempc{##1}%
2566   \def\bbl@tempb{##2}}%
2567 \expandafter\bbl@tempe\bbl@tempa..@@
2568 \bbl@csarg\edef{calpr@languagename}{%
2569   \ifx\bbl@tempc\@empty\else
2570     calendar=\bbl@tempc
2571   \fi
2572   \ifx\bbl@tempb\@empty\else
2573     ,variant=\bbl@tempb
2574   \fi}%
2575 % == engine specific extensions ==
2576 % Defined in XXXbabel.def
2577 \bbl@provide@extra{#2}%
2578 % == require.babel in ini ==
2579 % To load or reload the babel-*.tex, if require.babel in ini
2580 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2581   \bbl@ifunset{bbl@rqtex@languagename}{}%
2582   {\expandafter\ifx\csname bbl@rqtex@languagename\endcsname\@empty\else
2583     \let\BabelBeforeIni\@gobbletwo
2584     \chardef\atcatcode=\catcode`\<
2585     \catcode`\<=11\relax
2586     \def\CurrentOption{#2}%
2587     \bbl@input@texini{\bbl@cs{rqtex@languagename}}%
2588     \catcode`\<=\atcatcode
2589     \let\atcatcode\relax
2590     \global\bbl@csarg\let{rqtex@languagename}\relax
2591   \fi}%
2592 \bbl@foreach\bbl@calendars{%
2593   \bbl@ifunset{bbl@ca##1}{%
2594     \chardef\atcatcode=\catcode`\<
2595     \catcode`\<=11\relax
2596     \InputIfFileExists{babel-ca-##1.tex}{}}%
2597     \catcode`\<=\atcatcode
2598     \let\atcatcode\relax}%
2599   {}%
2600 \fi
2601 % == frenchspacing ==
2602 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2603 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2604 \ifin@
2605   \bbl@extras@wrap{\\bbl@pre@fs}%
2606   {\bbl@pre@fs}%
2607   {\bbl@post@fs}%
2608 \fi
2609 % == transforms ==
2610 % > luababel.def

```

```

2611 % == main ==
2612 \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2613   \let\language\bbbl@savelangname
2614   \chardef\localeid\bbbl@savelocaleid\relax
2615 \fi
2616 % == hyphenrules (apply if current) ==
2617 \ifx\bbbl@KVP@hyphenrules\@nnil\else
2618   \ifnum\bbbl@savelocaleid=\localeid
2619     \language\@nameuse{l@\language}%
2620   \fi
2621 \fi}

```

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

```

2622 \def\bbbl@provide@new#1{%
2623   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2624   \@namedef{extras#1}{}%
2625   \@namedef{noextras#1}{}%
2626   \bbbl@startcommands*{#1}{captions}%
2627   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2628     \def\bbbl@tempb##1{% elt for \bbbl@captionslist
2629       \ifx##1\@empty\else
2630         \bbbl@exp{%
2631           \\SetString\\##1{%
2632             \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2633         \expandafter\bbbl@tempb
2634       \fi}%
2635   \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2636 \else
2637   \ifx\bbbl@initoload\relax
2638     \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2639   \else
2640     \bbbl@read@ini{\bbbl@initoload}2% % Same
2641   \fi
2642 \fi
2643 \StartBabelCommands*{#1}{date}%
2644 \ifx\bbbl@KVP@date\@nnil
2645   \bbbl@exp{%
2646     \\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2647 \else
2648   \bbbl@savetoday
2649   \bbbl@savestate
2650 \fi
2651 \bbbl@endcommands
2652 \bbbl@load@basic{#1}%
2653 % == hyphenmins == (only if new)
2654 \bbbl@exp{%
2655   \gdef\<#1hyphenmins>{%
2656     {\bbbl@ifunset{\bbbl@lfthm#1}{2}{\bbbl@cs{lfthm#1}}}%
2657     {\bbbl@ifunset{\bbbl@rgthm#1}{3}{\bbbl@cs{rgthm#1}}}}}%
2658 % == hyphenrules (also in renew) ==
2659 \bbbl@provide@hyphens{#1}%
2660 \ifx\bbbl@KVP@main\@nnil\else
2661   \expandafter\main@language\expandafter{#1}%
2662 \fi}
2663 %
2664 \def\bbbl@provide@renew#1{%
2665   \ifx\bbbl@KVP@captions\@nnil\else
2666     \StartBabelCommands*{#1}{captions}%
2667     \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2668     \EndBabelCommands
2669   \fi
2670   \ifx\bbbl@KVP@date\@nnil\else

```

```

2671 \StartBabelCommands*{#1}{date}%
2672 \bbl@savetoday
2673 \bbl@savedate
2674 \EndBabelCommands
2675 \fi
2676 % == hyphenrules (also in new) ==
2677 \ifx\bbl@lbpflag\@empty
2678 \bbl@provide@hyphens{#1}%
2679 \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. (TODO. But preserving previous values would be useful.)

```

2680 \def\bbl@load@basic#1{%
2681 \ifcase\bbl@howloaded\or\or
2682 \ifcase\csname bbl@llevel@\language\endcsname
2683 \bbl@csarg\let\lname@\language\relax
2684 \fi
2685 \fi
2686 \bbl@ifunset{\bbl@lname@#1}%
2687 {\def\BabelBeforeIni##1##2{%
2688 \begingroup
2689 \let\bbl@ini@captions@aux\@gobbletwo
2690 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2691 \bbl@read@ini{##1}1%
2692 \ifx\bbl@initoload\relax\endinput\fi
2693 \endgroup}%
2694 \begingroup % boxed, to avoid extra spaces:
2695 \ifx\bbl@initoload\relax
2696 \bbl@input@texini{#1}%
2697 \else
2698 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2699 \fi
2700 \endgroup}%
2701 {}}

```

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.

```

2702 \def\bbl@provide@hyphens#1{%
2703 \@tempcnta\m@ne % a flag
2704 \ifx\bbl@KVP@hyphenrules\@nnil\else
2705 \bbl@replace\bbl@KVP@hyphenrules{ },}%
2706 \bbl@foreach\bbl@KVP@hyphenrules{%
2707 \ifnum\@tempcnta=\m@ne % if not yet found
2708 \bbl@ifsamestring{##1}{+}%
2709 {\bbl@carg\addlanguage{l@##1}}%
2710 {}}%
2711 \bbl@ifunset{l@##1}% After a possible +
2712 {}%
2713 {\@tempcnta\@nameuse{l@##1}}%
2714 \fi}%
2715 \ifnum\@tempcnta=\m@ne
2716 \bbl@warning{%
2717 Requested 'hyphenrules' for '\language' not found:\\%
2718 \bbl@KVP@hyphenrules.\\%
2719 Using the default value. Reported}%
2720 \fi
2721 \fi
2722 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2723 \ifx\bbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2724 \bbl@ifunset{\bbl@hyphr@#1}% use value in ini, if exists
2725 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2726 {}%
2727 {\bbl@ifunset{l@#1\bbl@cl{hyphr}}}%

```

```

2728          {}%                if hyphenrules found:
2729          {\@tempcnta\@nameuse{l@bbl@c{hyphr}}}}}%
2730      \fi
2731  \fi
2732  \bbl@ifunset{l@#1}%
2733      {\ifnum\@tempcnta=\m@ne
2734          \bbl@carg\adddialect{l@#1}\language
2735      \else
2736          \bbl@carg\adddialect{l@#1}\@tempcnta
2737      \fi}%
2738      {\ifnum\@tempcnta=\m@ne\else
2739          \global\bbl@carg\chardef{l@#1}\@tempcnta
2740      \fi}}

```

The reader of babel-...tex files. We reset temporarily some catcodes.

```

2741 \def\bbl@input@texini#1{%
2742     \bbl@bsphack
2743     \bbl@exp{%
2744         \catcode`\\%=14 \catcode`\\|=0
2745         \catcode`\\{=1 \catcode`\\}=2
2746         \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2747         \catcode`\\%=the\catcode`\\%\relax
2748         \catcode`\\|=the\catcode`\\|\relax
2749         \catcode`\\{=the\catcode`\\{\relax
2750         \catcode`\\}=the\catcode`\\}\relax}%
2751     \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.

```

2752 \def\bbl@iniline#1\bbl@iniline{%
2753     \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2754 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2755 \def\bbl@iniskip#1\@@{%          if starts with ;
2756 \def\bbl@inistore#1=#2\@@{%      full (default)
2757     \bbl@trim@def\bbl@tempa{#1}%
2758     \bbl@trim\toks@{#2}%
2759     \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2760     \ifin@else
2761         \bbl@xin@{,identification/include.}%
2762         {,\bbl@section/\bbl@tempa}%
2763     \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2764     \bbl@exp{%
2765         \\g@addto@macro\\bbl@inidata{%
2766             \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2767     \fi}
2768 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2769     \bbl@trim@def\bbl@tempa{#1}%
2770     \bbl@trim\toks@{#2}%
2771     \bbl@xin@{.identification.}{.\bbl@section.}%
2772     \ifin@
2773         \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2774             \\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2775     \fi}

```

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

```

2776 \def\bbl@loop@ini{%
2777     \loop
2778         \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop

```

```

2779 \endlinechar\m@ne
2780 \read\bb@l@readstream to \bb@l@line
2781 \endlinechar\^^M
2782 \ifx\bb@l@line\@empty\else
2783 \expandafter\bb@l@iniline\bb@l@line\bb@l@iniline
2784 \fi
2785 \repeat}
2786 \ifx\bb@l@readstream\@undefined
2787 \csname newread\endcsname\bb@l@readstream
2788 \fi
2789 \def\bb@l@read@ini#1#2{%
2790 \global\let\bb@l@extend@ini\@gobble
2791 \openin\bb@l@readstream=babel-#1.ini
2792 \ifeof\bb@l@readstream
2793 \bb@l@error{no-ini-file}{#1}{}}%
2794 \else
2795 % == Store ini data in \bb@l@inidata ==
2796 \catcode\|=12 \catcode\|=12 \catcode\|=12 \catcode\&=12
2797 \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2798 \bb@l@info{Importing
2799 \ifcase#2font and identification \or basic \fi
2800 data for \language\}%
2801 from babel-#1.ini. Reported}%
2802 \ifnum#2=\z@
2803 \global\let\bb@l@inidata\@empty
2804 \let\bb@l@inistore\bb@l@inistore@min % Remember it's local
2805 \fi
2806 \def\bb@l@section{identification}%
2807 \bb@l@exp{\bb@l@inistore tag.ini=#1\@@}%
2808 \bb@l@inistore load.level=#2\@@
2809 \bb@l@loop@ini
2810 % == Process stored data ==
2811 \bb@l@csarg\xdef\l@ini@language{#1}%
2812 \bb@l@read@ini@aux
2813 % == 'Export' data ==
2814 \bb@l@ini@exports{#2}%
2815 \global\bb@l@csarg\let\l@inidata@language\bb@l@inidata
2816 \global\let\bb@l@inidata\@empty
2817 \bb@l@exp{\bb@l@add@list\bb@l@ini@loaded{language}}%
2818 \bb@l@tglobal\bb@l@ini@loaded
2819 \fi
2820 \closein\bb@l@readstream}
2821 \def\bb@l@read@ini@aux{%
2822 \let\bb@l@savestrings\@empty
2823 \let\bb@l@savetoday\@empty
2824 \let\bb@l@savestate\@empty
2825 \def\bb@l@elt##1##2##3{%
2826 \def\bb@l@section{##1}%
2827 \in@{=date.}{=##1}% Find a better place
2828 \ifin@
2829 \bb@l@ifunset{bb@l@inikv@##1}%
2830 {\bb@l@ini@calendar{##1}}%
2831 }%
2832 \fi
2833 \bb@l@ifunset{bb@l@inikv@##1}{}%
2834 {\csname bb@l@inikv@##1\endcsname{##2}{##3}}%
2835 \bb@l@inidata}

```

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

```

2836 \def\bb@l@extend@ini@aux#1{%
2837 \bb@l@startcommands*{#1}{captions}%
2838 % Activate captions/... and modify exports

```



```

2839 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2840 \setlocalecaption{#1}{##1}{##2}}%
2841 \def\bbl@inikv@captions##1##2{%
2842 \bbl@ini@captions@aux{##1}{##2}}%
2843 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2844 \def\bbl@exportkey##1##2##3{%
2845 \bbl@ifunset{bbl@kv@##2}{}%
2846 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2847 \bbl@exp{\global\let<bbl@##1@language>\<bbl@kv@##2>}}%
2848 \fi}%
2849 % As with \bbl@read@ini, but with some changes
2850 \bbl@read@ini@aux
2851 \bbl@ini@exports\tw@
2852 % Update inidata@lang by pretending the ini is read.
2853 \def\bbl@elt##1##2##3{%
2854 \def\bbl@section{##1}%
2855 \bbl@iniline##2=##3\bbl@iniline}%
2856 \csname bbl@inidata@#1\endcsname
2857 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2858 \StartBabelCommands*{#1}{date}% And from the import stuff
2859 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2860 \bbl@savetoday
2861 \bbl@savestate
2862 \bbl@endcommands}

```

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

```

2863 \def\bbl@ini@calendar#1{%
2864 \lowercase{\def\bbl@tempa{#1=}}%
2865 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2866 \bbl@replace\bbl@tempa{=date.}{}%
2867 \in@{.licr}={#1=}%
2868 \ifin@
2869 \ifcase\bbl@engine
2870 \bbl@replace\bbl@tempa{.licr}={}%
2871 \else
2872 \let\bbl@tempa\relax
2873 \fi
2874 \fi
2875 \ifx\bbl@tempa\relax\else
2876 \bbl@replace\bbl@tempa{=}{}%
2877 \ifx\bbl@tempa\@empty\else
2878 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2879 \fi
2880 \bbl@exp{%
2881 \def<bbl@inikv@#1>####1####2{%
2882 \\bbl@inidata####1...\relax{####2}{\bbl@tempa}}}%
2883 \fi}

```

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

```

2884 \def\bbl@renewinikey#1/#2\@#3{%
2885 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2886 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2887 \bbl@trim\toks@{#3}% value
2888 \bbl@exp{%
2889 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2890 \\g@addto@macro\\bbl@inidata{%
2891 \\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

2892 \def\bbl@exportkey#1#2#3{%

```

```

2893 \bbl@ifunset{bbl@kv@#2}%
2894   {\bbl@csarg\gdef{#1@language}{#3}}%
2895   {\expandafter\ifx\csname bbl@kv@#2\endcsname\@empty
2896     \bbl@csarg\gdef{#1@language}{#3}}%
2897   \else
2898     \bbl@exp{\global\let\<bbl@#1@language>\<bbl@kv@#2>}%
2899   \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note `\bbl@ini@exports` is called always (via `\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary. Although BCP 47 doesn't treat 'x-' as an extension, the CLDR and many other sources do (as a *private use extension*). For consistency with other single-letter subtags or 'singletons', here is considered an extension, too.

```

2900 \def\bbl@iniwarning#1{%
2901   \bbl@ifunset{bbl@kv@identification.warning#1}{}%
2902   {\bbl@warning{%
2903     From babel-\bbl@cs{lini@language}.ini:\%
2904     \bbl@cs{kv@identification.warning#1}\%
2905     Reported }}%
2906 %
2907 \let\bbl@release@transforms\@empty
2908 \let\bbl@release@casing\@empty
2909 \def\bbl@ini@exports#1{%
2910   % Identification always exported
2911   \bbl@iniwarning{}%
2912   \ifcase\bbl@engine
2913     \bbl@iniwarning{.pdflatex}%
2914   \or
2915     \bbl@iniwarning{.lua\latex}%
2916   \or
2917     \bbl@iniwarning{.xel\latex}%
2918   \fi%
2919   \bbl@exportkey{llevel}{identification.load.level}{}%
2920   \bbl@exportkey{elname}{identification.name.english}{}%
2921   \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2922     {\csname bbl@elname@language\endcsname}}%
2923   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2924   % Somewhat hackish. TODO:
2925   \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2926   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
2927   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2928   \bbl@exportkey{esname}{identification.script.name}{}%
2929   \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2930     {\csname bbl@esname@language\endcsname}}%
2931   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2932   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2933   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2934   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2935   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2936   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2937   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2938   % Also maps bcp47 -> language
2939   \ifbbl@bcptoname
2940     \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
2941   \fi
2942   \ifcase\bbl@engine\or
2943     \directlua{%
2944       Babel.locale_props[\the\bbl@cs{id@language}].script
2945       = '\bbl@cl{sbc}}}%
2946   \fi
2947   % Conditional
2948   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new

```

```

2949 \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2950 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2951 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2952 \bbl@exportkey{lftm}{typography.leftthyphenmin}{2}%
2953 \bbl@exportkey{rgthm}{typography.righththyphenmin}{3}%
2954 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2955 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2956 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2957 \bbl@exportkey{intsp}{typography.intraspacespace}{}%
2958 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2959 \bbl@exportkey{chrng}{characters.ranges}{}%
2960 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2961 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2962 \ifnum#1=\tw@ % only (re)new
2963 \bbl@exportkey{rqtex}{identification.require.babel}{}%
2964 \bbl@toglobal\bbl@savetoday
2965 \bbl@toglobal\bbl@savestate
2966 \bbl@savestrings
2967 \fi
2968 \fi}

```

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

```

2969 \def\bbl@inikv#1#2{%      key=value
2970 \toks@{#2}%              This hides #'s from ini values
2971 \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}

```

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

```

2972 \let\bbl@inikv@identification\bbl@inikv
2973 \let\bbl@inikv@date\bbl@inikv
2974 \let\bbl@inikv@typography\bbl@inikv
2975 \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.

```

2976 \def\bbl@maybextx{-\bbl@csarg\ifx{extx@\language}\empty x-\fi}
2977 \def\bbl@inikv@characters#1#2{%
2978 \bbl@ifsamestring{#1}{casing}% eg, casing = uV
2979 {\bbl@exp{%
2980 \\\g@addto@macro\\\bbl@release@casing{%
2981 \\\bbl@casemapping}{\language}\unexpanded{#2}}}%
2982 {\in@{casing.}{#1}% eg, casing.Uv = uV
2983 \ifin@
2984 \lowercase{\def\bbl@tempb{#1}}%
2985 \bbl@replace\bbl@tempb{casing.}{}%
2986 \bbl@exp{\\g@addto@macro\\\bbl@release@casing{%
2987 \\\bbl@casemapping
2988 {\\\bbl@maybextx\bbl@tempb}{\language}\unexpanded{#2}}}%
2989 \else
2990 \bbl@inikv{#1}{#2}%
2991 \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’.

```

2992 \def\bbl@inikv@counters#1#2{%
2993 \bbl@ifsamestring{#1}{digits}%
2994 {\bbl@error{digits-is-reserved}{}}}%
2995 {}%
2996 \def\bbl@tempc{#1}%
2997 \bbl@trim@def{\bbl@tempb*}{#2}%
2998 \in@{.1$}{#1$}%
2999 \ifin@
3000 \bbl@replace\bbl@tempc{.1}{}%

```

```

3001 \bbl@csarg\protected@xdef{cntr@bbl@tempc @\language}\language}{%
3002 \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3003 \fi
3004 \in@{.F.}{#1}%
3005 \ifin@else\in@{.S.}{#1}\fi
3006 \ifin@
3007 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3008 \else
3009 \toks@{} Required by \bbl@buildifcase, which returns \bbl@tempa
3010 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3011 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3012 \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.

```

3013 \ifcase\bbl@engine
3014 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3015 \bbl@ini@captions@aux{#1}{#2}}
3016 \else
3017 \def\bbl@inikv@captions#1#2{%
3018 \bbl@ini@captions@aux{#1}{#2}}
3019 \fi

```

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

```

3020 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3021 \bbl@replace\bbl@tempa{.template}}%
3022 \def\bbl@toreplace{#1}%
3023 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3024 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3025 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3026 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}%
3027 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3028 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3029 \ifin@
3030 \@nameuse{\bbl@patch\bbl@tempa}%
3031 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3032 \fi
3033 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3034 \ifin@
3035 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3036 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3037 \\\bbl@ifunset{\bbl@\bbl@tempa fmt@\\\language}%
3038 {[fnum@\bbl@tempa]}%
3039 {\\\@nameuse{\bbl@\bbl@tempa fmt@\\\language}}}}%
3040 \fi}
3041 \def\bbl@ini@captions@aux#1#2{%
3042 \bbl@trim@def\bbl@tempa{#1}%
3043 \bbl@xin@{.template}{\bbl@tempa}%
3044 \ifin@
3045 \bbl@ini@captions@template{#2}\language
3046 \else
3047 \bbl@ifblank{#2}%
3048 {\bbl@exp{%
3049 \toks@{\\\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3050 {\bbl@trim\toks@{#2}}}%
3051 \bbl@exp{%
3052 \\\bbl@add\\bbl@savestrings{%
3053 \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3054 \toks@\expandafter{\bbl@captionslist}%
3055 \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3056 \ifin@else
3057 \bbl@exp{%
3058 \\\bbl@add\<bbl@extracaps@\language>{\<\bbl@tempa name>}}%

```

```

3059      \\bbl@toglobal\<bbl@extracaps@\language>%
3060      \fi
3061      \fi}

```

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

```

3062 \def\bbl@list@the{%
3063   part,chapter,section,subsection,subsubsection,paragraph,%
3064   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3065   table,page,footnote,mpfootnote,mpfn}
3066 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3067   \bbl@ifunset{bbl@map@#1@\language}%
3068     {\@nameuse{#1}}%
3069     {\@nameuse{bbl@map@#1@\language}}}
3070 \def\bbl@inikv@labels#1#2{%
3071   \in@{.map}{#1}%
3072   \ifin@
3073     \ifx\bbl@KVP@labels\@nnil\else
3074       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3075       \ifin@
3076         \def\bbl@tempc{#1}%
3077         \bbl@replace\bbl@tempc{.map}{}%
3078         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3079         \bbl@exp{%
3080           \gdef\<bbl@map@\bbl@tempc @\language>%
3081             {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3082         \bbl@foreach\bbl@list@the{%
3083           \bbl@ifunset{the##1}{}%
3084           {\bbl@exp{\let\\bbl@tempd\<the##1>%
3085             \bbl@exp{%
3086               \\bbl@sreplace\<the##1>%
3087               {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3088               \\bbl@sreplace\<the##1>%
3089               {\<\@empty @\bbl@tempc>\<c@##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3090             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3091               \toks@ \expandafter\expandafter\expandafter{%
3092                 \csname the##1\endcsname}%
3093               \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%
3094             \fi}}%
3095         \fi
3096       \fi
3097     %
3098   \else
3099     %
3100     % The following code is still under study. You can test it and make
3101     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3102     % language dependent.
3103     \in@{enumerate.}{#1}%
3104     \ifin@
3105       \def\bbl@tempa{#1}%
3106       \bbl@replace\bbl@tempa{enumerate.}{}%
3107       \def\bbl@toreplace{#2}%
3108       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3109       \bbl@replace\bbl@toreplace{[]}{\csname the}%
3110       \bbl@replace\bbl@toreplace{[]}{\endcsname}}%
3111       \toks@ \expandafter{\bbl@toreplace}%
3112       % TODO. Execute only once:
3113       \bbl@exp{%
3114         \\bbl@add\<extras\language>%
3115         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3116         \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3117         \\bbl@toglobal\<extras\language>%
3118       \fi
3119     \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.

```

3120 \def\bbbl@chapttype{chapter}
3121 \ifx\@makechapterhead\@undefined
3122   \let\bbbl@patchchapter\relax
3123 \else\ifx\thechapter\@undefined
3124   \let\bbbl@patchchapter\relax
3125 \else\ifx\ps@headings\@undefined
3126   \let\bbbl@patchchapter\relax
3127 \else
3128   \def\bbbl@patchchapter{%
3129     \global\let\bbbl@patchchapter\relax
3130     \gdef\bbbl@chfmt{%
3131       \bbbl@ifunset{bbbl@bbbl@chapttype fmt@\language}%
3132       {\@chapapp\space\thechapter}
3133       {\@nameuse{bbbl@bbbl@chapttype fmt@\language}}}
3134     \bbbl@add\appendix{\def\bbbl@chapttype{appendix}}% Not harmful, I hope
3135     \bbbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbbl@chfmt}%
3136     \bbbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbbl@chfmt}%
3137     \bbbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbbl@chfmt}%
3138     \bbbl@tglobal\appendix
3139     \bbbl@tglobal\ps@headings
3140     \bbbl@tglobal\chaptermark
3141     \bbbl@tglobal\@makechapterhead}
3142   \let\bbbl@patchappendix\bbbl@patchchapter
3143 \fi\fi\fi
3144 \ifx\@part\@undefined
3145   \let\bbbl@patchpart\relax
3146 \else
3147   \def\bbbl@patchpart{%
3148     \global\let\bbbl@patchpart\relax
3149     \gdef\bbbl@partformat{%
3150       \bbbl@ifunset{bbbl@partfmt@\language}%
3151       {\partname\nobreakspace\thepart}
3152       {\@nameuse{bbbl@partfmt@\language}}}
3153     \bbbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbbl@partformat}%
3154     \bbbl@tglobal\@part}
3155 \fi

```

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

```

3156 \let\bbbl@calendar\@empty
3157 \DeclareRobustCommand\localedate[1][\bbbl@localedate{#1}]
3158 \def\bbbl@localedate#1#2#3#4{%
3159   \begingroup
3160     \edef\bbbl@they{#2}%
3161     \edef\bbbl@them{#3}%
3162     \edef\bbbl@thed{#4}%
3163     \edef\bbbl@tempe{%
3164       \bbbl@ifunset{bbbl@calpr@\language}{\bbbl@cl{calpr}},%
3165       #1}%
3166     \bbbl@replace\bbbl@tempe{ }{}%
3167     \bbbl@replace\bbbl@tempe{CONVERT}{convert}% Hackish
3168     \bbbl@replace\bbbl@tempe{convert}{convert}%
3169     \let\bbbl@ld@calendar\@empty
3170     \let\bbbl@ld@variant\@empty
3171     \let\bbbl@ld@convert\relax
3172     \def\bbbl@tempb##1=##2\@{\@namedef{bbbl@ld@##1}{##2}}%
3173     \bbbl@foreach\bbbl@tempe{\bbbl@tempb##1\@}%
3174     \bbbl@replace\bbbl@ld@calendar{gregorian}{}%
3175     \ifx\bbbl@ld@calendar\@empty\else

```

```

3176 \ifx\babel@ld@convert\relax\else
3177 \babelcalendar[\babel@they-\babel@them-\babel@thed]%
3178 {\babel@ld@calendar}\babel@they\babel@them\babel@thed
3179 \fi
3180 \fi
3181 \@nameuse{babel@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3182 \edef\babel@calendar{% Used in \month..., too
3183 \babel@ld@calendar
3184 \ifx\babel@ld@variant\empty\else
3185 .\babel@ld@variant
3186 \fi}%
3187 \babel@cased
3188 {\@nameuse{babel@date@\language name @\babel@calendar}%
3189 \babel@they\babel@them\babel@thed}%
3190 \endgroup}
3191 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3192 \def\babel@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3193 \babel@trim@def\babel@tempa{#1.#2}%
3194 \babel@ifsamestring{\babel@tempa}{months.wide}% to savedate
3195 {\babel@trim@def\babel@tempa{#3}%
3196 \babel@trim\toks@{#5}%
3197 \@temptokena\expandafter{\babel@savestate}%
3198 \babel@exp{% Reverse order - in ini last wins
3199 \def\babel@savestate{%
3200 \SetString<month\romannumeral\babel@tempa#6name>{\the\toks@}%
3201 \the\@temptokena}}}%
3202 {\babel@ifsamestring{\babel@tempa}{date.long}% defined now
3203 {\lowercase{\def\babel@tempb{#6}}}%
3204 \babel@trim@def\babel@toreplace{#5}%
3205 \babel@TG@@date
3206 \global\babel@csarg\let{date@\language name @\babel@tempb}\babel@toreplace
3207 \ifx\babel@savestate\empty
3208 \babel@exp{% TODO. Move to a better place.
3209 \AfterBabelCommands{%
3210 \def<\language name date>{\protect<\language name date >}%
3211 \newcommand<\language name date >[4][{}%
3212 \babel@usedategroupttrue
3213 <\babel@ensure@\language name>{%
3214 \localdate[####1]{####2}{####3}{####4}}}%
3215 \def\babel@savestate{%
3216 \SetString\babel@today{%
3217 <\language name date>[convert]%
3218 {\the\year}{\the\month}{\the\day}}}%
3219 \fi}%
3220 {}}}}

```

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 `\babel@replace\toks@` contains the resulting string, which is used by `\babel@replace@finish@iii` (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3221 \let\babel@calendar\empty
3222 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3223 \@nameuse{babel@ca#2}#1\@{
3224 \newcommand\BabelDateSpace{\nobreakspace}
3225 \newcommand\BabelDateDot{\. \@ % TODO. \let instead of repeating
3226 \newcommand\BabelDated[1]{\number#1}}
3227 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}}
3228 \newcommand\BabelDateM[1]{\number#1}}
3229 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}}
3230 \newcommand\BabelDateMMM[1]{%
3231 \csname month\romannumeral#1\babel@calendar name\endcsname}%
3232 \newcommand\BabelDatey[1]{\number#1}%

```

```

3233 \newcommand\BabelDateyy[1]{%
3234   \ifnum#1<10 0\number#1 %
3235   \else\ifnum#1<100 \number#1 %
3236   \else\ifnum#1<1000 \expandafter@gobble\number#1 %
3237   \else\ifnum#1<10000 \expandafter@gobbletwo\number#1 %
3238   \else
3239     \bbl@error{limit-two-digits}{\}\}%
3240   \fi\fi\fi\fi}}
3241 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3242 \newcommand\BabelDateU[1]{\number#1}%
3243 \def\bbl@replace@finish@iii#1{%
3244   \bbl@exp{\def\#1###1####2####3{\the\toks@}}
3245 \def\bbl@TG@date{%
3246   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3247   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3248   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3249   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3250   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3251   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3252   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3253   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3254   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3255   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3256   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{####1}}%
3257   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecntr{####1}}%
3258   \bbl@replace\bbl@toreplace{[U]}{\bbl@datecntr{####1}}%
3259   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecntr{####2}}%
3260   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecntr{####3}}%
3261   \bbl@replace@finish@iii\bbl@toreplace}
3262 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3263 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3264 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3265 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3266 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3267   #1[#2]{#3}{#4}{#5}}
3268 \begingroup % A hack. TODO. Don't require an specific order
3269   \catcode`\%=12
3270   \catcode`\&=14
3271   \gdef\bbl@transforms#1#2#3{&%
3272     \directlua{
3273       local str = [=[#2]=]
3274       str = str:gsub('%.%d+%.%d+$', '')
3275       token.set_macro('babeltempa', str)
3276     }&%
3277     \def\babeltempc{}&%
3278     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3279     \ifin@ \else
3280       \bbl@xin@{: \babeltempa,}{,\bbl@KVP@transforms,}&%
3281     \fi
3282     \ifin@
3283       \bbl@foreach\bbl@KVP@transforms{&%
3284         \bbl@xin@{: \babeltempa,}{,##1,}&%
3285         \ifin@ &% font:font:transform syntax
3286         \directlua{
3287           local t = {}
3288           for m in string.gmatch('##1'..' ':'(.)') do
3289             table.insert(t, m)
3290           end
3291           table.remove(t)
3292           token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3293         }&%

```



```

3294     \fi}%&%
3295     \in@{.0$}{#2$}%&%
3296     \ifin@
3297         \directlua{%& (\attribute) syntax
3298             local str = string.match([[ \bbl@KVP@transforms]],
3299                 '%(([^%()-)]^%)]-\babeltempa')
3300             if str == nil then
3301                 token.set_macro('babeltempb', '')
3302             else
3303                 token.set_macro('babeltempb', ',attribute=' .. str)
3304             end
3305         }&%
3306     \toks@{#3}%&%
3307     \bbl@exp{%&%
3308         \\g@addto@macro\\bbl@release@transforms{%&%
3309             \relax &% Closes previous \bbl@transforms@aux
3310             \\bbl@transforms@aux
3311             \\#1{label=\babeltempa\babeltempb\babeltempc}%&%
3312             {\language\the\toks@}}}%&%
3313     \else
3314         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3315     \fi
3316 \fi}
3317 \endgroup

```

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

```

3318 \def\bbl@provide@lsys#1{%
3319     \bbl@ifunset{bbl@lname@#1}%
3320     {\bbl@load@info{#1}}%
3321     {}%
3322     \bbl@csarg\let{lsys@#1}\@empty
3323     \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3324     \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3325     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3326     \bbl@ifunset{bbl@lname@#1}{}%
3327     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3328     \ifcase\bbl@engine\or\or
3329         \bbl@ifunset{bbl@prehc@#1}{}%
3330         {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3331             {}%
3332             {\ifx\bbl@xenohyph\@undefined
3333                 \global\let\bbl@xenohyph\bbl@xenohyph@d
3334                 \ifx\AtBeginDocument\@notprerr
3335                     \expandafter\@secondoftwo % to execute right now
3336                 \fi
3337                 \AtBeginDocument{%
3338                     \bbl@patchfont{\bbl@xenohyph}%
3339                     {\expandafter\select@language\expandafter{\language\the\toks@}}}%
3340             \fi}}%
3341     \fi
3342     \bbl@csarg\bbl@tglobal{lsys@#1}}
3343 \def\bbl@xenohyph@d{%
3344     \bbl@ifset{bbl@prehc@language}%
3345     {\ifnum\hyphenchar\font=\defaultshyphenchar
3346         \iffontchar\font\bbl@cl{prehc}\relax
3347         \hyphenchar\font\bbl@cl{prehc}\relax
3348     \else\iffontchar\font"200B
3349         \hyphenchar\font"200B
3350     \else
3351         \bbl@warning
3352         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3353         in the current font, and therefore the hyphen\\%

```

```

3354         will be printed. Try changing the fontspec's\\%
3355         'HyphenChar' to another value, but be aware\\%
3356         this setting is not safe (see the manual).\\%
3357         Reported}%
3358         \hyphenchar\font\defaultthyphenchar
3359         \fi\fi
3360         \fi}%
3361         {\hyphenchar\font\defaultthyphenchar}}
3362     % \fi}

```

```

3363 \def\bbl@load@info#1{%
3364   \def\BabelBeforeIni##1##2{%
3365     \begingroup
3366       \bbl@read@ini{##1}0%
3367       \endinput           % babel- .tex may contain onlypreamble's
3368     \endgroup}%          boxed, to avoid extra spaces:
3369   {\bbl@input@texini{#1}}}
```

[illegible]

```

3401 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3402   \ifx\##1%                % \ before, in case #1 is multiletter
3403     \bbl@exp{%
3404       \def\\bbl@tempa####1{%
3405         \ifcase>####1\space\the\toks@\<else>\\@ctrerrr<fi>}}%
3406   \else

```

```

3407 \toks@expandafter{\the\toks@or #1}%
3408 \expandafter\bb@buildifcase
3409 \fi}

```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before @@ collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey .F., the number after is treated as an special case, for a fixed form (see babel-he.ini, for example).

```

3410 \newcommand\localecnumeral[2]{\bb@cs{cntr@#1@language}{#2}}
3411 \def\bb@localecntr#1#2{\localecnumeral{#2}{#1}}
3412 \newcommand\localecounter[2]{%
3413 \expandafter\bb@localecntr
3414 \expandafter{\number\csname c@#2\endcsname}{#1}}
3415 \def\bb@alphanumeric#1#2{%
3416 \expandafter\bb@alphanumeric@i\number#2 76543210\@@{#1}}
3417 \def\bb@alphanumeric@i#1#2#3#4#5#6#7#8\@@#9{%
3418 \ifcase\car#8\@nil\or % Currently <10000, but prepared for bigger
3419 \bb@alphanumeric@ii{#9}00000#1\or
3420 \bb@alphanumeric@ii{#9}00000#1#2\or
3421 \bb@alphanumeric@ii{#9}00000#1#2#3\or
3422 \bb@alphanumeric@ii{#9}000#1#2#3#4\else
3423 \bb@alphanum@invalid{>9999}%
3424 \fi}
3425 \def\bb@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3426 \bb@ifunset{bb@cntr@#1.F.\number#5#6#7#8@language}%
3427 {\bb@cs{cntr@#1.4@language}{#5}
3428 \bb@cs{cntr@#1.3@language}{#6}
3429 \bb@cs{cntr@#1.2@language}{#7}
3430 \bb@cs{cntr@#1.1@language}{#8}
3431 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3432 \bb@ifunset{bb@cntr@#1.S.321@language}{}%
3433 {\bb@cs{cntr@#1.S.321@language}{}}%
3434 \fi}%
3435 {\bb@cs{cntr@#1.F.\number#5#6#7#8@language}}%
3436 \def\bb@alphanum@invalid#1{%
3437 \bb@error{alphabetic-too-large}{#1}{}}%

```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```

3438 \def\bb@localeinfo#1#2{%
3439 \bb@ifunset{bb@info@#2}{#1}%
3440 {\bb@ifunset{bb@\csname bbl@info@#2\endcsname @language}{#1}%
3441 {\bb@cs{\csname bbl@info@#2\endcsname @language}}}%
3442 \newcommand\localeinfo[1]{%
3443 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3444 \bb@afterelse\bb@localeinfo}%
3445 \else
3446 \bb@localeinfo
3447 {\bb@error{no-ini-info}{}}}%
3448 {#1}%
3449 \fi}
3450 % \@namedef{bb@info@name.locale}{lcname}
3451 \@namedef{bb@info@tag.ini}{lini}
3452 \@namedef{bb@info@name.english}{elname}
3453 \@namedef{bb@info@name.opentype}{lname}
3454 \@namedef{bb@info@tag.bcp47}{tbc}
3455 \@namedef{bb@info@language.tag.bcp47}{lbc}
3456 \@namedef{bb@info@tag.opentype}{lotf}
3457 \@namedef{bb@info@script.name}{esname}
3458 \@namedef{bb@info@script.name.opentype}{sname}
3459 \@namedef{bb@info@script.tag.bcp47}{sbcp}
3460 \@namedef{bb@info@script.tag.opentype}{sotf}
3461 \@namedef{bb@info@region.tag.bcp47}{rbcp}

```

```

3462 \@namedef{bbl@info@variant.tag.bcp47}{vbc}
3463 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3464 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3465 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}

```

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

```

3466 \ifcase\bbl@engine % Converts utf8 to its code (expandable)
3467   \def\bbl@uftocode#1{\the\numexpr\decode@UTFviii#1\relax}
3468 \else
3469   \def\bbl@uftocode#1{\expandafter`\string#1}
3470 \fi
3471 % Still somewhat hackish. WIP. Note |\str_if_eq:nnTF| is fully
3472 % expandable (|\bbl@ifsamestring| isn't).
3473 \providecommand\BCPdata{}
3474 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3475   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3476   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3477     \@nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3478     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3479     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\languagename}}%
3480   \def\bbl@bcpdata@ii#1#2{%
3481     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3482     {\bbl@error{unknown-ini-field}{#1}}{}%
3483     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3484      {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3485 \fi
3486 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3487 \newcommand\BabelUppercaseMapping[3]{%
3488   \DeclareUppercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3489 \newcommand\BabelTitlecaseMapping[3]{%
3490   \DeclareTitlecaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}
3491 \newcommand\BabelLowercaseMapping[3]{%
3492   \DeclareLowercaseMapping[\@nameuse{bbl@casing@#1}]{#2}{#3}}

```

The parser for casing and casing. (*variant*).

```

3493 \def\bbl@casemapping#1#2#3{% 1:variant
3494   \def\bbl@tempa##1 ##2{% Loop
3495     \bbl@casemapping@i{##1}%
3496     \ifx\@empty##2\else\bbl@afterfi\bbl@tempa##2\fi}%
3497   \edef\bbl@templ{\@nameuse{bbl@casing@#2}#1}% Language code
3498   \def\bbl@tempe{0}% Mode (upper/lower...)
3499   \def\bbl@tempc{#3}% Casing list
3500   \expandafter\bbl@tempa\bbl@tempc\@empty}
3501 \def\bbl@casemapping@i#1{%
3502   \def\bbl@tempb{#1}%
3503   \ifcase\bbl@engine % Handle utf8 in pdftex, by surrounding chars with {}
3504     \@nameuse{regex_replace_all:nnN}%
3505     {[\\x{c0}-\\x{ff}][\\x{80}-\\x{bf}]*}{\0}}\bbl@tempb
3506   \else
3507     \@nameuse{regex_replace_all:nnN}{.}{\0}}\bbl@tempb % TODO. needed?
3508   \fi
3509   \expandafter\bbl@casemapping@ii\bbl@tempb@@}
3510 \def\bbl@casemapping@ii#1#2#3\@@{%
3511   \in@{#1#3}{<>}% ie, if <u>, <l>, <t>
3512   \ifin@
3513     \edef\bbl@tempe{%
3514       \if#2u1 \else\if#2l2 \else\if#2t3 \fi\fi\fi}%
3515   \else
3516     \ifcase\bbl@tempe\relax
3517       \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3518       \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#2}}{#1}%
3519     \or

```

```

3520 \DeclareUppercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3521 \or
3522 \DeclareLowercaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3523 \or
3524 \DeclareTitlecaseMapping[\bbl@templ]{\bbl@uftocode{#1}}{#2}%
3525 \fi
3526 \fi}

```

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

```

3527 <<{*More package options}>> ≡
3528 \DeclareOption{ensureinfo=off}{}
3529 <</More package options>>
3530 \let\bbl@ensureinfo@\gobble
3531 \newcommand\BabelEnsureInfo{%
3532   \ifx\InputIfFileExists\@undefined\else
3533     \def\bbl@ensureinfo##1{%
3534       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3535   \fi
3536   \bbl@foreach\bbl@loaded{{%
3537     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3538     \def\language{##1}%
3539     \bbl@ensureinfo{##1}}}
3540 \@ifpackagewith{babel}{ensureinfo=off}{}%
3541 {\AtEndOfPackage{% Test for plain.
3542   \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}

```

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

```

3543 \newcommand\getlocaleproperty{%
3544   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3545 \def\bbl@getproperty@s#1#2#3{%
3546   \let#1\relax
3547   \def\bbl@elt##1##2##3{%
3548     \bbl@ifsamestring{##1/##2}{#3}%
3549     {\providecommand#1{##3}%
3550     \def\bbl@elt####1####2####3{}}}%
3551   {}}%
3552   \bbl@cs{inidata@#2}}%
3553 \def\bbl@getproperty@x#1#2#3{%
3554   \bbl@getproperty@s{#1}{#2}{#3}%
3555   \ifx#1\relax
3556     \bbl@error{unknown-locale-key}{#1}{#2}{#3}%
3557   \fi}
3558 \let\bbl@ini@loaded\@empty
3559 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
3560 \def\ShowLocaleProperties#1{%
3561   \typeout{}%
3562   \typeout{*** Properties for language '#1' ***}
3563   \def\bbl@elt##1##2##3{\typeout{##1/##2 = ##3}}%
3564   \@nameuse{bbl@inidata@#1}%
3565   \typeout{*****}}

```

5 Adjusting the Babel behavior

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

```

3566 \newcommand\babeladjust[1]{% TODO. Error handling.
3567   \bbl@forkv{#1}{%
3568     \bbl@ifunset{bbl@ADJ@##1@##2}%
3569     {\bbl@cs{ADJ@##1}{##2}}%
3570     {\bbl@cs{ADJ@##1@##2}}}
3571 %

```

```

3572 \def\bbl@adjust@lua#1#2{%
3573   \ifvmode
3574     \ifnum\currentgrouplevel=\z@
3575       \directlua{ Babel.#2 }%
3576       \expandafter\expandafter\expandafter\@gobble
3577     \fi
3578   \fi
3579   {\bbl@error{adjust-only-vertical}{#1}{}}}% Gobbled if everything went ok.
3580 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3581   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3582 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3583   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3584 \@namedef{bbl@ADJ@bidi.text@on}{%
3585   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3586 \@namedef{bbl@ADJ@bidi.text@off}{%
3587   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3588 \@namedef{bbl@ADJ@bidi.math@on}{%
3589   \let\bbl@noamsmath\@empty}
3590 \@namedef{bbl@ADJ@bidi.math@off}{%
3591   \let\bbl@noamsmath\relax}
3592 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3593   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3594 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3595   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3596 %
3597 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3598   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3599 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3600   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3601 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3602   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3603 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3604   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3605 \@namedef{bbl@ADJ@justify.arabic@on}{%
3606   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3607 \@namedef{bbl@ADJ@justify.arabic@off}{%
3608   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3609 %
3610 \def\bbl@adjust@layout#1{%
3611   \ifvmode
3612     #1%
3613     \expandafter\@gobble
3614   \fi
3615   {\bbl@error{layout-only-vertical}{}}}% Gobbled if everything went ok.
3616 \@namedef{bbl@ADJ@layout.tabular@on}{%
3617   \ifnum\bbl@tabular@mode=\tw@
3618     \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3619   \else
3620     \chardef\bbl@tabular@mode\@ne
3621   \fi}
3622 \@namedef{bbl@ADJ@layout.tabular@off}{%
3623   \ifnum\bbl@tabular@mode=\tw@
3624     \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}%
3625   \else
3626     \chardef\bbl@tabular@mode\z@
3627   \fi}
3628 \@namedef{bbl@ADJ@layout.lists@on}{%
3629   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3630 \@namedef{bbl@ADJ@layout.lists@off}{%
3631   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3632 %
3633 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3634   \bbl@bcpallowedtrue}

```

```

3635 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3636   \bbl@bcpallowedfalse}
3637 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3638   \def\bbl@bcp@prefix{#1}}
3639 \def\bbl@bcp@prefix{bcp47-}
3640 \@namedef{bbl@ADJ@autoload.options}#1{%
3641   \def\bbl@autoload@options{#1}}
3642 \let\bbl@autoload@bcptoptions\@empty
3643 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3644   \def\bbl@autoload@bcptoptions{#1}}
3645 \newif\ifbbl@bcptoname
3646 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3647   \bbl@bcptonametrue
3648   \BabelEnsureInfo}
3649 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3650   \bbl@bcptonamefalse}
3651 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3652   \directlua{ Babel.ignore_pre_char = function(node)
3653     return (node.lang == \the\csname l@nohyphenation\endcsname)
3654   end }}
3655 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3656   \directlua{ Babel.ignore_pre_char = function(node)
3657     return false
3658   end }}
3659 \@namedef{bbl@ADJ@select.write@shift}{%
3660   \let\bbl@restorelastskip\relax
3661   \def\bbl@savelastskip{%
3662     \let\bbl@restorelastskip\relax
3663     \ifvmode
3664       \ifdim\lastskip=\z@
3665         \let\bbl@restorelastskip\nobreak
3666       \else
3667         \bbl@exp{%
3668           \def\\bbl@restorelastskip{%
3669             \skip@=\the\lastskip
3670             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3671       \fi
3672     \fi}}
3673 \@namedef{bbl@ADJ@select.write@keep}{%
3674   \let\bbl@restorelastskip\relax
3675   \let\bbl@savelastskip\relax}
3676 \@namedef{bbl@ADJ@select.write@omit}{%
3677   \AddBabelHook{babel-select}{beforestart}{%
3678     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3679 \let\bbl@restorelastskip\relax
3680 \def\bbl@savelastskip##1\bbl@restorelastskip{}
3681 \@namedef{bbl@ADJ@select.encoding@off}{%
3682   \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.

```

3683 <(*More package options)> ≡
3684 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}

```

```

3685 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3686 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3687 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3688 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3689 <</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.

```

3690 \bbl@trace{Cross referencing macros}
3691 \ifx\bbl@opt@safe\empty\else % ie, if 'ref' and/or 'bib'
3692   \def\@newl@bel#1#2#3{%
3693     {\@safe@activestrue
3694       \bbl@ifunset{#1#2}%
3695         \relax
3696         {\gdef\@multiplelabels{%
3697           \@latex@warning@no@line{There were multiply-defined labels}}%
3698           \@latex@warning@no@line{Label `#2' multiply defined}}%
3699       \global\@namedef{#1#2}{#3}}

```

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

```

3700 \CheckCommand*\@testdef[3]{%
3701   \def\reserved@a{#3}%
3702   \expandafter\ifx\csname#1#2\endcsname\reserved@a
3703   \else
3704     \@tempswatrue
3705   \fi}

```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands ‘safe’. Then we use `\bbl@tempa` as an ‘alias’ for the macro that contains the label which is being checked. Then we define `\bbl@tempb` just as `\@newl@bel` does it. When the label is defined we replace the definition of `\bbl@tempa` by its meaning. If the label didn’t change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

```

3706 \def\@testdef#1#2#3{% TODO. With @samestring?
3707   \@safe@activestrue
3708   \expandafter\let\expandafter\bbl@tempa\csname #1#2\endcsname
3709   \def\bbl@tempb{#3}%
3710   \@safe@activesfalse
3711   \ifx\bbl@tempa\relax
3712   \else
3713     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3714   \fi
3715   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3716   \ifx\bbl@tempa\bbl@tempb
3717   \else
3718     \@tempswatrue
3719   \fi}
3720 \fi

```

`\ref` 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.

```

3721 \bbl@xin@{R}\bbl@opt@safe
3722 \ifin@
3723   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3724   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3725   {\expandafter\strip@prefix\meaning\ref}%
3726 \ifin@
3727   \bbl@redefine\@kernel@ref#1{%
3728     \@safe@activestrue\org@kernel@ref{#1}\@safe@activesfalse}
3729   \bbl@redefine\@kernel@pageref#1{%
3730     \@safe@activestrue\org@kernel@pageref{#1}\@safe@activesfalse}

```



```

3731 \bbl@redefine\@kernel@sref#1{%
3732 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activfalse}
3733 \bbl@redefine\@kernel@spageref#1{%
3734 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activfalse}
3735 \else
3736 \bbl@redefineroast\ref#1{%
3737 \@safe@activetrue\org@ref{#1}\@safe@activfalse}
3738 \bbl@redefineroast\pageref#1{%
3739 \@safe@activetrue\org@pageref{#1}\@safe@activfalse}
3740 \fi
3741 \else
3742 \let\org@ref\ref
3743 \let\org@pageref\pageref
3744 \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.

```

3745 \bbl@xin@{B}\bbl@opt@safe
3746 \ifin@
3747 \bbl@redefine\@citex[#1]#2{%
3748 \@safe@activetrue\edef\bbl@tempa{#2}\@safe@activfalse
3749 \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.

```

3750 \AtBeginDocument{%
3751 \ifpackageloaded{natbib}{%

```

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.)

```

3752 \def\@citex[#1][#2]#3{%
3753 \@safe@activetrue\edef\bbl@tempa{#3}\@safe@activfalse
3754 \org@@citex[#1][#2]{\bbl@tempa}}%
3755 }{}

```

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

```

3756 \AtBeginDocument{%
3757 \ifpackageloaded{cite}{%
3758 \def\@citex[#1]#2{%
3759 \@safe@activetrue\org@@citex[#1][#2]\@safe@activfalse}%
3760 }{}

```

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

```

3761 \bbl@redefine\nocite#1{%
3762 \@safe@activetrue\org@nocite{#1}\@safe@activfalse}

```

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

```

3763 \bbl@redefine\bibcite{%
3764 \bbl@cite@choice
3765 \bibcite}

```

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

```
3766 \def\bbl@bibtex#1#2{%
3767   \org@bibtex{#1}\@safe@activesfalse#2}}
```

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

```
3768 \def\bbl@cite@choice{%
3769   \global\let\bibtex\bbl@bibtex
3770   \ifpackageloaded{natbib}\global\let\bibtex\org@bibtex}%
3771   \ifpackageloaded{cite}\global\let\bibtex\org@bibtex}%
3772   \global\let\bbl@cite@choice\relax}
```

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

```
3773 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3774 \bbl@redefine\@bibitem#1{%
3775   \@safe@activestrue\org@bibitem{#1}\@safe@activesfalse}
3776 \else
3777   \let\org@nocite\nocite
3778   \let\org@citem\citem
3779   \let\org@bibtex\bibtex
3780   \let\org@bibitem\@bibitem
3781 \fi
```

5.2 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.

```
3782 \bbl@trace{Marks}
3783 \IfBabelLayout{sectioning}
3784   {\ifx\bbl@opt@headfoot\@nnil
3785     \g@addto@macro\@resetactivechars{%
3786       \set@typeset@protect
3787       \expandafter\select@language\expandafter{\bbl@main@language}%
3788       \let\protect\noexpand
3789       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3790         \edef\thepage{%
3791           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3792       \fi}%
3793   \fi}
3794 {\ifbbl@single\else
3795   \bbl@ifunset{markright }\bbl@redefine\bbl@redefineroobust
3796   \markright#1{%
3797     \bbl@ifblank{#1}%
3798     {\org@markright{}}%
3799     {\toks@{#1}%
3800      \bbl@exp{%
3801        \\\org@markright{\protect\\foreignlanguage{\language}\language}%
3802        {\protect\\bbl@restore@actives\the\toks@}}}%
3803   }
```

`\markboth` 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, \LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3803 \ifx\@mkboth\markboth
3804 \def\bb@tempc{\let\@mkboth\markboth}%
3805 \else
3806 \def\bb@tempc{%
3807 \fi
3808 \bb@ifunset{markboth }\bb@redefine\bb@redefineroobust
3809 \markboth#1#2{%
3810 \protected@edef\bb@tempb##1{%
3811 \protect\foreignlanguage
3812 {\language\name}{\protect\bb@restore@actives##1}}%
3813 \bb@ifblank{#1}%
3814 {\toks@{}}%
3815 {\toks@\expandafter{\bb@tempb{#1}}}%
3816 \bb@ifblank{#2}%
3817 {\@temptokena{}}%
3818 {\@temptokena\expandafter{\bb@tempb{#2}}}%
3819 \bb@exp{\org@markboth{\the\toks@}{\the\@temptokena}}}%
3820 \bb@tempc
3821 \fi} % end ifbb@single, end \IfBabelLayout

```

5.3 Preventing clashes with other packages

5.3.1 `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.

```

3822 \bb@trace{Preventing clashes with other packages}
3823 \ifx\org@ref\undefined\else
3824 \bb@xin@{R}\bb@opt@safe
3825 \ifin@
3826 \AtBeginDocument{%
3827 \ifpackageloaded{ifthen}{%
3828 \bb@redefine@long\ifthenelse#1#2#3{%
3829 \let\bb@temp@pref\pageref
3830 \let\pageref\org@pageref
3831 \let\bb@temp@ref\ref
3832 \let\ref\org@ref
3833 \@safe@activestrue
3834 \org@ifthenelse{#1}%
3835 {\let\pageref\bb@temp@pref
3836 \let\ref\bb@temp@ref
3837 \@safe@activesfalse
3838 #2}%
3839 {\let\pageref\bb@temp@pref
3840 \let\ref\bb@temp@ref
3841 \@safe@activesfalse
3842 #3}%

```

```

3843         }%
3844     }{}%
3845 }
3846 \fi

```

5.3.2 varioref

`\@@vpageref` When the package `varioref` is in use we need to modify its internal command `\@@vpageref` in order to prevent problems when an active character ends up in the argument of `\vref`. The same needs to happen for `\vrefpagenum`.

```

3847 \AtBeginDocument{%
3848     \ifpackageloaded{varioref}{%
3849         \bbl@redefine\@@vpageref#1[#2]#3{%
3850             \@safe@activetrue
3851             \org@@vpageref{#1}[#2]{#3}%
3852             \@safe@activesfalse}%
3853         \bbl@redefine\vrefpagenum#1#2{%
3854             \@safe@activetrue
3855             \org\vrefpagenum{#1}{#2}%
3856             \@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.

```

3857     \expandafter\def\csname Ref \endcsname#1{%
3858         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3859     }{}%
3860 }
3861 \fi

```

5.3.3 hhline

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

```

3862 \AtEndOfPackage{%
3863     \AtBeginDocument{%
3864         \ifpackageloaded{hhline}%
3865         {\expandafter\ifx\csname normal@char\string\endcsname\relax
3866             \else
3867                 \makeatletter
3868                 \def\@currname{hhline}\input{hhline.sty}\makeatother
3869             \fi}%
3870         {}}}

```

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

```

3871 \def\substitutefontfamily#1#2#3{%
3872     \lowercase{\immediate\openout15=#1#2.fd\relax}%
3873     \immediate\write15{%
3874         \string\ProvidesFile{#1#2.fd}%
3875         [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3876         \space generated font description file]^{}
3877         \string\DeclareFontFamily{#1}{#2}{}}^{}
3878         \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^{}
3879         \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^{}
3880         \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^{}

```

```

3881 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^J
3882 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^J
3883 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^J
3884 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^J
3885 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^J
3886 }%
3887 \closeout15
3888 }
3889 \@onlypreamble\substitutefontfamily

```

5.4 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`

```

3890 \bbl@trace{Encoding and fonts}
3891 \newcommand\BabelNonASCII{LGR,LGI,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3892 \newcommand\BabelNonText{TS1,T3,TS3}
3893 \let\org@TeX\TeX
3894 \let\org@LaTeX\LaTeX
3895 \let\ensureascii\@firstofone
3896 \let\asciientcoding\@empty
3897 \AtBeginDocument{%
3898   \def\@elt#1{,#1,}%
3899   \edef\bbl@tempa{\expandafter\@gobbletwo\fontenc@load@list}%
3900   \let\@elt\relax
3901   \let\bbl@tempb\@empty
3902   \def\bbl@tempc{OT1}%
3903   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3904     \bbl@ifunset{T@#1}{ }\def\bbl@tempb{#1}}}%
3905   \bbl@foreach\bbl@tempa{%
3906     \bbl@xin@{,#1,}{,\BabelNonASCII,}%
3907     \ifin@
3908       \def\bbl@tempb{#1}% Store last non-ascii
3909     \else\bbl@xin@{,#1,}{,\BabelNonText,}% Pass
3910       \ifin@\else
3911         \def\bbl@tempc{#1}% Store last ascii
3912       \fi
3913     \fi}%
3914   \ifx\bbl@tempb\@empty\else
3915     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3916     \ifin@\else
3917       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3918     \fi
3919     \let\asciientcoding\bbl@tempc
3920     \renewcommand\ensureascii[1]{%
3921       {\fontencoding{\asciientcoding}\selectfont#1}}%
3922     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3923     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3924   \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.

```

3925 \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.

```

3926 \AtBeginDocument{%
3927   \ifpackageloaded{fontspec}%
3928     {\xdef\latinencoding{%
3929       \ifx\UTFencname\@undefined
3930         EU\ifcase\bb@engine\or2\or1\fi
3931       \else
3932         \UTFencname
3933       \fi}}%
3934   {\gdef\latinencoding{OT1}%
3935     \ifx\cf@encoding\bb@t@one
3936       \xdef\latinencoding{\bb@t@one}%
3937     \else
3938       \def\@elt#1{,#1,%}
3939       \edef\bb@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3940       \let\@elt\relax
3941       \bb@xin@{,T1,}\bb@tempa
3942       \ifin@
3943         \xdef\latinencoding{\bb@t@one}%
3944       \fi
3945     \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.

```

3946 \DeclareRobustCommand{\latintext}{%
3947   \fontencoding{\latinencoding}\selectfont
3948   \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.

```

3949 \ifx\@undefined\DeclareTextFontCommand
3950   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3951 \else
3952   \DeclareTextFontCommand{\textlatin}{\latintext}
3953 \fi

```

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

```

3954 \def\bb@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.5 Basic bidi support

Work in progress. 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 LuaT_X-ja shows, vertical typesetting is possible, too.

```

3955 \bbl@trace{Loading basic (internal) bidi support}
3956 \ifodd\bbl@engine
3957 \else % TODO. Move to txtbabel
3958   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3959     \bbl@error{bidi-only-lua}{}}{}%
3960     \let\bbl@beforeforeign\leavevmode
3961     \AtEndOfPackage{%
3962       \EnableBabelHook{babel-bidi}%
3963       \bbl@xebidipar}
3964   \fi\fi
3965   \def\bbl@loadxebidi#1{%
3966     \ifx\RTLfootnotetext\@undefined
3967       \AtEndOfPackage{%
3968         \EnableBabelHook{babel-bidi}%
3969         \bbl@loadfontspec % bidi needs fontspec
3970         \usepackage#1{bidi}%
3971         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
3972         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
3973           \ifnum\@nameuse{bbl@wdir@\languagename}=\tw@ % 'AL' bidi
3974             \bbl@digitsdotdash % So ignore in 'R' bidi
3975           \fi}%
3976         \fi}
3977   \ifnum\bbl@bidimode>200 % Any xe bidi=
3978     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3979       \bbl@tentative{bidi=bidi}
3980       \bbl@loadxebidi{}
3981     \or
3982       \bbl@loadxebidi{[rldocument]}
3983     \or
3984       \bbl@loadxebidi{}
3985     \fi
3986   \fi
3987 \fi
3988 % TODO? Separate:
3989 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
3990   \let\bbl@beforeforeign\leavevmode
3991   \ifodd\bbl@engine
3992     \newattribute\bbl@attr@dir
3993     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3994     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3995   \fi
3996   \AtEndOfPackage{%
3997     \EnableBabelHook{babel-bidi}%
3998     \ifodd\bbl@engine\else
3999       \bbl@xebidipar
4000     \fi}
4001 \fi

```

Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.

```

4002 \bbl@trace{Macros to switch the text direction}
4003 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4004 \def\bbl@rscripts{% TODO. Base on codes ??
4005   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4006   Old Hungarian,Lydian,Mandaean,Manichaeen,%
4007   Meroitic Cursive,Meroitic,Old North Arabian,%
4008   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4009   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4010   Old South Arabian,}%

```

```

4011 \def\bbl@provide@dirs#1{%
4012   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4013   \ifin@
4014     \global\bbl@csarg\chardef{wdir@#1}\@ne
4015     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4016     \ifin@
4017       \global\bbl@csarg\chardef{wdir@#1}\tw@
4018     \fi
4019   \else
4020     \global\bbl@csarg\chardef{wdir@#1}\z@
4021   \fi
4022   \ifodd\bbl@engine
4023     \bbl@csarg\ifcase{wdir@#1}%
4024       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4025     \or
4026       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4027     \or
4028       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4029     \fi
4030   \fi}
4031 \def\bbl@switchdir{%
4032   \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4033   \bbl@ifunset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4034   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
4035 \def\bbl@setdirs#1{% TODO - math
4036   \ifcase\bbl@select@type % TODO - strictly, not the right test
4037     \bbl@bodydir{#1}%
4038     \bbl@pardir{#1}% <- Must precede \bbl@texdir
4039   \fi
4040   \bbl@texdir{#1}}
4041 % TODO. Only if \bbl@bidimode > 0?:
4042 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4043 \DisableBabelHook{babel-bidi}

```

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

```

4044 \ifodd\bbl@engine % luatex=1
4045 \else % pdftex=0, xetex=2
4046   \newcount\bbl@dirlevel
4047   \chardef\bbl@thetexdir\z@
4048   \chardef\bbl@thepardir\z@
4049   \def\bbl@texdir#1{%
4050     \ifcase#1\relax
4051       \chardef\bbl@thetexdir\z@
4052       \@nameuse{setlatin}%
4053       \bbl@texdir@i\beginL\endL
4054     \else
4055       \chardef\bbl@thetexdir\@ne
4056       \@nameuse{setnonlatin}%
4057       \bbl@texdir@i\beginR\endR
4058     \fi}
4059   \def\bbl@texdir@i#1#2{%
4060     \ifhmode
4061       \ifnum\currentgrouplevel>\z@
4062         \ifnum\currentgrouplevel=\bbl@dirlevel
4063           \bbl@error{multiple-bidi}{}{}%
4064           \bgroup\aftergroup#2\aftergroup\egroup
4065         \else
4066           \ifcase\currentgrouptype\or % 0 bottom
4067             \aftergroup#2 % 1 simple {}
4068           \or
4069             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4070           \or
4071             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox

```



```

4072      \or\or\or % vbox vtop align
4073      \or
4074      \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4075      \or\or\or\or\or\or % output math disc insert vcent mathchoice
4076      \or
4077      \aftergroup#2% 14 \begingroup
4078      \else
4079      \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4080      \fi
4081      \fi
4082      \bbl@dirlevel\currentgrouplevel
4083      \fi
4084      #1%
4085      \fi}
4086      \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4087      \let\bbl@bodydir\@gobble
4088      \let\bbl@pagedir\@gobble
4089      \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).

```

4090      \def\bbl@xebidipar{%
4091      \let\bbl@xebidipar\relax
4092      \TeXeTstate\@ne
4093      \def\bbl@xeeverypar{%
4094      \ifcase\bbl@thepardir
4095      \ifcase\bbl@thetextdir\else\beginR\fi
4096      \else
4097      {\setbox\z@\lastbox\beginR\box\z@}%
4098      \fi}%
4099      \let\bbl@severypar\everypar
4100      \newtoks\everypar
4101      \everypar=\bbl@severypar
4102      \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4103      \ifnum\bbl@bidimode>200 % Any xe bidi=
4104      \let\bbl@textdir@i\@gobbletwo
4105      \let\bbl@xebidipar\@empty
4106      \AddBabelHook{bidi}{foreign}{%
4107      \def\bbl@tempa{\def\BabelText###1}%
4108      \ifcase\bbl@thetextdir
4109      \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4110      \else
4111      \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4112      \fi}
4113      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4114      \fi
4115      \fi

```

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

```

4116      \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4117      \AtBeginDocument{%
4118      \ifx\pdfstringdefDisableCommands\@undefined\else
4119      \ifx\pdfstringdefDisableCommands\relax\else
4120      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4121      \fi
4122      \fi}

```

5.6 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`.

```

4123 \bbl@trace{Local Language Configuration}
4124 \ifx\loadlocalcfg\undefined
4125   \@ifpackagewith{babel}{noconfigs}%
4126   {\let\loadlocalcfg\gobble}%
4127   {\def\loadlocalcfg#1{%
4128     \InputIfFileExists{#1.cfg}%
4129     {\typeout{*****^J}%
4130      * Local config file #1.cfg used^^J%
4131      *}}%
4132     \@empty}}
4133 \fi

```

5.7 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).

```

4134 \bbl@trace{Language options}
4135 \let\bbl@afterlang\relax
4136 \let\BabelModifiers\relax
4137 \let\bbl@loaded\@empty
4138 \def\bbl@load@language#1{%
4139   \InputIfFileExists{#1.ldf}%
4140   {\edef\bbl@loaded{\CurrentOption
4141     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4142     \expandafter\let\expandafter\bbl@afterlang
4143       \csname\CurrentOption.ldf-h@k\endcsname
4144     \expandafter\let\expandafter\BabelModifiers
4145       \csname bbl@mod@\CurrentOption\endcsname
4146     \bbl@exp{\AtBeginDocument{%
4147       \bbl@usehooks@lang{\CurrentOption}{\begin{document}}{\CurrentOption}}}%
4148     {\IfFileExists{babel-#1.tex}%
4149      {\def\bbl@tempa{%
4150        .\There is a locale ini file for this language.\%
4151        If it's the main language, try adding `provide=''\%
4152        to the babel package options}}%
4153      {\let\bbl@tempa\empty}%
4154      \bbl@error{unknown-package-option}{\CurrentOption}}}%

```

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

```

4155 \def\bbl@try@load@lang#1#2#3{%
4156   \IfFileExists{\CurrentOption.ldf}%
4157   {\bbl@load@language{\CurrentOption}}%
4158   {\bbl@load@language{#2}#3}}
4159 %
4160 \DeclareOption{hebrew}{%
4161   \ifcase\bbl@engine\or
4162     \bbl@error{only-pdftex-lang}{hebrew}{luatex}}%
4163 \fi
4164 \input{rlbabel.def}%
4165 \bbl@load@language{hebrew}}
4166 \DeclareOption{hungarian}{\bbl@try@load@lang{\magyar}}%
4167 \DeclareOption{lowersorbian}{\bbl@try@load@lang{\lsorbian}}%
4168 \DeclareOption{polutonikogreek}{%
4169   \bbl@try@load@lang{\greek}\languageattribute{greek}{polutoniko}}%
4170 \DeclareOption{russian}{\bbl@try@load@lang{\russianb}}%
4171 \DeclareOption{ukrainian}{\bbl@try@load@lang{\ukraineb}}%
4172 \DeclareOption{uppersorbian}{\bbl@try@load@lang{\usorbian}}%

```

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

```

4173 \ifx\bblopt@config\@nnil
4174 \ifpackagewith{babel}{noconfigs}{}%
4175   {\InputIfFileExists{bblopts.cfg}%
4176     {\typeout{*****^J%
4177               * Local config file bblopts.cfg used^J%
4178               *}}%
4179     }{}%
4180 \else
4181 \InputIfFileExists{\bblopt@config.cfg}%
4182   {\typeout{*****^J%
4183             * Local config file \bblopt@config.cfg used^J%
4184             *}}%
4185   {\bblopt@error{config-not-found}{}}{}%
4186 \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 `bblopt@language@opts` are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the main language, which is processed in the third ‘main’ pass, *except* if all files are `ldf` and there is no main key. In the latter case (`\bblopt@main` is still `\@nnil`), the traditional way to set the main language is kept — the last loaded is the main language.

```

4187 \ifx\bblopt@main\@nnil
4188 \ifnum\bblopt@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4189   \let\bblopt@tempb\@empty
4190   \edef\bblopt@tempa{\@classoptionslist,\bblopt@language@opts}%
4191   \bblopt@foreach\bblopt@tempa{\edef\bblopt@tempb{#1,\bblopt@tempb}}%
4192   \bblopt@foreach\bblopt@tempb{% \bblopt@tempb is a reversed list
4193     \ifx\bblopt@main\@nnil % ie, if not yet assigned
4194       \ifodd\bblopt@iniflag % = *=
4195         \IfFileExists{babel-#1.tex}{\def\bblopt@main{#1}}{}%
4196       \else % n +=
4197         \IfFileExists{#1.ldf}{\def\bblopt@main{#1}}{}%
4198       \fi
4199     \fi}%
4200 \fi
4201 \else
4202 \bblopt@info{Main language set with 'main='. Except if you have\\%
4203             problems, prefer the default mechanism for setting\\%
4204             the main language, ie, as the last declared.\\%
4205             Reported}%
4206 \fi

```

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

```

4207 \ifx\bblopt@main\@nnil\else
4208 \bblopt@ncarg\let\bblopt@loadmain{ds@\bblopt@main}%
4209 \expandafter\let\csname ds@\bblopt@main\endcsname\relax
4210 \fi

```

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

```

4211 \bblopt@foreach\bblopt@language@opts{%
4212   \def\bblopt@tempa{#1}%
4213   \ifx\bblopt@tempa\bblopt@opt@main\else
4214     \ifnum\bblopt@iniflag<\tw@ % 0 0 (other = ldf)
4215       \bblopt@ifunset{ds@#1}%
4216       {\DeclareOption{#1}{\bblopt@load@language{#1}}}%
4217     {}%

```

```

4218 \else % + * (other = ini)
4219 \DeclareOption{#1}{%
4220 \bbl@ldfinit
4221 \babelprovide[import]{#1}%
4222 \bbl@afterldf{}}%
4223 \fi
4224 \fi}
4225 \bbl@foreach\@classoptionslist{%
4226 \def\bbl@tempa{#1}%
4227 \ifx\bbl@tempa\bbl@opt@main\else
4228 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4229 \bbl@ifunset{ds@#1}%
4230 {\IfFileExists{#1.ldf}%
4231 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4232 {}}%
4233 {}}%
4234 \else % + * (other = ini)
4235 \IfFileExists{babel-#1.tex}%
4236 {\DeclareOption{#1}{%
4237 \bbl@ldfinit
4238 \babelprovide[import]{#1}%
4239 \bbl@afterldf{}}}%
4240 {}}%
4241 \fi
4242 \fi}

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

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

```

4243 \def\AfterBabelLanguage#1{%
4244 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4245 \DeclareOption*{}
4246 \ProcessOptions*

```

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

```

4247 \bbl@trace{Option 'main'}
4248 \ifx\bbl@opt@main\@nnil
4249 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4250 \let\bbl@tempc\@empty
4251 \edef\bbl@templ{\bbl@loaded,}
4252 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4253 \bbl@for\bbl@tempb\bbl@tempa{%
4254 \edef\bbl@tempd{\bbl@tempb,}%
4255 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4256 \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4257 \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4258 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4259 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4260 \ifx\bbl@tempb\bbl@tempc\else
4261 \bbl@warning{%
4262 Last declared language option is '\bbl@tempc',\\%
4263 but the last processed one was '\bbl@tempb'.\\%
4264 The main language can't be set as both a global\\%
4265 and a package option. Use 'main=\bbl@tempc' as\\%
4266 option. Reported}
4267 \fi
4268 \else
4269 \ifodd\bbl@iniflag % case 1,3 (main is ini)

```

```

4270 \bbl@ldfinit
4271 \let\CurrentOption\bbl@opt@main
4272 \bbl@exp{% \bbl@opt@provide = empty if *
4273 \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4274 \bbl@afterldf{}
4275 \DeclareOption{\bbl@opt@main}{}
4276 \else % case 0,2 (main is ldf)
4277 \ifx\bbl@loadmain\relax
4278 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4279 \else
4280 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4281 \fi
4282 \ExecuteOptions{\bbl@opt@main}
4283 \@namedef{ds@\bbl@opt@main}{}%
4284 \fi
4285 \DeclareOption*{}
4286 \ProcessOptions*
4287 \fi
4288 \bbl@exp{%
4289 \\\AtBeginDocument{\bbl@usehooks@lang{/}{begindocument}{}}}%
4290 \def\AfterBabelLanguage{\bbl@error{late-after-babel}{}}{}

```

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

```

4291 \ifx\bbl@main@language\undefined
4292 \bbl@info{%
4293 You haven't specified a language as a class or package\\%
4294 option. I'll load 'nil'. Reported}
4295 \bbl@load@language{nil}
4296 \fi
4297 \end{package}

```

6 The kernel of Babel (`babel.def`, common)

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 $\text{T}_{\text{E}}\text{X}$ users might want to use some of the features of the babel system too, care has to be taken that plain $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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`

```

4298 \kernel
4299 \let\bbl@onlyswitch\@empty
4300 \input babel.def
4301 \let\bbl@onlyswitch\@undefined
4302 \end{kernel}
4303 %
4304 % \section{Error messages}
4305 %
4306 % They are loaded when |\bbl@error| is first called. To save space, the
4307 % main code just identifies them with a tag, and messages are stored in
4308 % a separate file. Since it can be loaded anywhere, you make sure some
4309 % catcodes have the right value, although those for |\|, |`|, |^M|,
4310 % |%| and |=| are reset before loading the file.
4311 %
4312 \errors
4313 \catcode`\{=1 \catcode`\}=2 \catcode`\#=6
4314 \catcode`\:=12 \catcode`\,=12 \catcode`\.=12 \catcode`\-=12

```

```

4315 \catcode`\'=12 \catcode`\(=12 \catcode`\)=12
4316 \catcode`\@=11 \catcode`\^=7
4317 %
4318 \ifx\MessageBreak\undefined
4319   \gdef\bbl@error@i#1#2{%
4320     \begingroup
4321       \newlinechar=\^^J
4322       \def\{\^^J(babel) }%
4323       \errhelp{#2}\errmessage{\#1}%
4324     \endgroup}
4325 \else
4326   \gdef\bbl@error@i#1#2{%
4327     \begingroup
4328       \def\{\MessageBreak}%
4329       \PackageError{babel}{#1}{#2}%
4330     \endgroup}
4331 \fi
4332 \def\bbl@errmessage#1#2#3{%
4333   \expandafter\gdef\csname bbl@err@#1\endcsname##1##2##3{%
4334     \bbl@error@i{#2}{#3}}}
4335 % Implicit #2#3#4:
4336 \gdef\bbl@error#1{\csname bbl@err@#1\endcsname}
4337 %
4338 \bbl@errmessage{not-yet-available}
4339   {Not yet available}%
4340   {Find an armchair, sit down and wait}
4341 \bbl@errmessage{bad-package-option}%
4342   {Bad option '#1=#2'. Either you have misspelled the\\%
4343     key or there is a previous setting of '#1'. Valid\\%
4344     keys are, among others, 'shorthands', 'main', 'bidi',\\%
4345     'strings', 'config', 'headfoot', 'safe', 'math'.}%
4346   {See the manual for further details.}
4347 \bbl@errmessage{base-on-the-fly}
4348   {For a language to be defined on the fly 'base'\\%
4349     is not enough, and the whole package must be\\%
4350     loaded. Either delete the 'base' option or\\%
4351     request the languages explicitly}%
4352   {See the manual for further details.}
4353 \bbl@errmessage{undefined-language}
4354   {You haven't defined the language '#1' yet.\\%
4355     Perhaps you misspelled it or your installation\\%
4356     is not complete}%
4357   {Your command will be ignored, type <return> to proceed}
4358 \bbl@errmessage{shorthand-is-off}
4359   {I can't declare a shorthand turned off (\string#2)}
4360   {Sorry, but you can't use shorthands which have been\\%
4361     turned off in the package options}
4362 \bbl@errmessage{not-a-shorthand}
4363   {The character '\string #1' should be made a shorthand character;\\%
4364     add the command \string\usesshorthands\string{#1\string} to
4365     the preamble.\\%
4366     I will ignore your instruction}%
4367   {You may proceed, but expect unexpected results}
4368 \bbl@errmessage{not-a-shorthand-b}
4369   {I can't switch '\string#2' on or off--not a shorthand}%
4370   {This character is not a shorthand. Maybe you made\\%
4371     a typing mistake? I will ignore your instruction.}
4372 \bbl@errmessage{unknown-attribute}
4373   {The attribute #2 is unknown for language #1.}%
4374   {Your command will be ignored, type <return> to proceed}
4375 \bbl@errmessage{missing-group}
4376   {Missing group for string \string#1}%
4377   {You must assign strings to some category, typically\\%

```

```

4378     captions or extras, but you set none}
4379 \bbl@errmessage{only-lua-xe}
4380     {This macro is available only in LuaLaTeX and XeLaTeX.}%
4381     {Consider switching to these engines.}
4382 \bbl@errmessage{only-lua}
4383     {This macro is available only in LuaLaTeX.}%
4384     {Consider switching to that engine.}
4385 \bbl@errmessage{unknown-provide-key}
4386     {Unknown key '#1' in \string\babelprovide}%
4387     {See the manual for valid keys}%
4388 \bbl@errmessage{unknown-mapfont}
4389     {Option '\bbl@KVP@mapfont' unknown for\\%
4390     mapfont. Use 'direction'.}%
4391     {See the manual for details.}
4392 \bbl@errmessage{no-ini-file}
4393     {There is no ini file for the requested language\\%
4394     (#1: \language). Perhaps you misspelled it or your\\%
4395     installation is not complete.}%
4396     {Fix the name or reinstall babel.}
4397 \bbl@errmessage{digits-is-reserved}
4398     {The counter name 'digits' is reserved for mapping\\%
4399     decimal digits}%
4400     {Use another name.}
4401 \bbl@errmessage{limit-two-digits}
4402     {Currently two-digit years are restricted to the\\
4403     range 0-9999.}%
4404     {There is little you can do. Sorry.}
4405 \bbl@errmessage{alphabetic-too-large}
4406     {Alphabetic numeral too large (#1)}%
4407     {Currently this is the limit.}
4408 \bbl@errmessage{no-ini-info}
4409     {I've found no info for the current locale.\\%
4410     The corresponding ini file has not been loaded\\%
4411     Perhaps it doesn't exist}%
4412     {See the manual for details.}
4413 \bbl@errmessage{unknown-ini-field}
4414     {Unknown field '#1' in \string\BCPdata.\\%
4415     Perhaps you misspelled it.}%
4416     {See the manual for details.}
4417 \bbl@errmessage{unknown-locale-key}
4418     {Unknown key for locale '#2':\\%
4419     #3\\%
4420     \string#1 will be set to \relax}%
4421     {Perhaps you misspelled it.}%
4422 \bbl@errmessage{adjust-only-vertical}
4423     {Currently, #1 related features can be adjusted only\\%
4424     in the main vertical list.}%
4425     {Maybe things change in the future, but this is what it is.}
4426 \bbl@errmessage{layout-only-vertical}
4427     {Currently, layout related features can be adjusted only\\%
4428     in vertical mode.}%
4429     {Maybe things change in the future, but this is what it is.}
4430 \bbl@errmessage{bidi-only-lua}
4431     {The bidi method 'basic' is available only in\\%
4432     luatex. I'll continue with 'bidi=default', so\\%
4433     expect wrong results}%
4434     {See the manual for further details.}
4435 \bbl@errmessage{multiple-bidi}
4436     {Multiple bidi settings inside a group}%
4437     {I'll insert a new group, but expect wrong results.}
4438 \bbl@errmessage{unknown-package-option}
4439     {Unknown option '\CurrentOption'. Either you misspelled it\\%
4440     or the language definition file \CurrentOption.ldf\\%

```

```

4441     was not found%
4442     \bbl@tempa}
4443     {Valid options are, among others: shorthands=, KeepShorthandsActive,\%
4444     activeacute, activegrave, noconfigs, safe=, main=, math=\%
4445     headfoot=, strings=, config=, hyphenmap=, or a language name.}
4446 \bbl@errmessage{config-not-found}
4447 {Local config file '\bbl@opt@config.cfg' not found}%
4448 {Perhaps you misspelled it.}
4449 \bbl@errmessage{late-after-babel}
4450 {Too late for \string\AfterBabelLanguage}%
4451 {Languages have been loaded, so I can do nothing}
4452 \bbl@errmessage{double-hyphens-class}
4453 {Double hyphens aren't allowed in \string\babelcharclass\%
4454     because it's potentially ambiguous}%
4455 {See the manual for further info}
4456 \bbl@errmessage{unknown-interchar}
4457 {'#1' for '\language' cannot be enabled.\%
4458     Maybe there is a typo.}%
4459 {See the manual for further details.}
4460 \bbl@errmessage{unknown-interchar-b}
4461 {'#1' for '\language' cannot be disabled.\%
4462     Maybe there is a typo.}%
4463 {See the manual for further details.}
4464 \bbl@errmessage{charproperty-only-vertical}
4465 {\string\babelcharproperty\space can be used only in\%
4466     vertical mode (preamble or between paragraphs)}%
4467 {See the manual for further info}
4468 \bbl@errmessage{unknown-char-property}
4469 {No property named '#2'. Allowed values are\%
4470     direction (bc), mirror (bmg), and linebreak (lb)}%
4471 {See the manual for further info}
4472 \bbl@errmessage{bad-transform-option}
4473 {Bad option '#1' in a transform.\%
4474     I'll ignore it but expect more errors}%
4475 {See the manual for further info.}
4476 \bbl@errmessage{font-conflict-transforms}
4477 {Transforms cannot be re-assigned to different\%
4478     fonts. The conflict is in '\bbl@kv@label'.\%
4479     Apply the same fonts or use a different label}%
4480 {See the manual for further details.}
4481 \bbl@errmessage{transform-not-available}
4482 {'#1' for '\language' cannot be enabled.\%
4483     Maybe there is a typo or it's a font-dependent transform}%
4484 {See the manual for further details.}
4485 \bbl@errmessage{transform-not-available-b}
4486 {'#1' for '\language' cannot be disabled.\%
4487     Maybe there is a typo or it's a font-dependent transform}%
4488 {See the manual for further details.}
4489 \bbl@errmessage{year-out-range}
4490 {Year out of range.\%
4491     The allowed range is #1}%
4492 {See the manual for further details.}
4493 \bbl@errmessage{only-pdftex-lang}
4494 {The '#1' ldf style doesn't work with #2,\%
4495     but you can use the ini locale instead.\%
4496     Try adding 'provide=*' to the option list. You may\%
4497     also want to set 'bidi=' to some value.}%
4498 {See the manual for further details.}
4499 \errors}
4500 \patterns}

```


7 Loading hyphenation patterns

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

```
4501 <<Make sure ProvidesFile is defined>>
4502 \ProvidesFile{hyphen.cfg}[\<date> v\<version>] Babel hyphens]
4503 \xdef\bbl@format{\jobname}
4504 \def\bbl@version{\<version>}
4505 \def\bbl@date{\<date>}
4506 \ifx\AtBeginDocument\undefined
4507   \def\@empty{}
4508 \fi
4509 <<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.

```
4510 \def\process@line#1#2 #3 #4 {%
4511   \ifx=#1%
4512     \process@synonym{#2}%
4513   \else
4514     \process@language{#1#2}{#3}{#4}%
4515   \fi
4516   \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.

```
4517 \toks@{}
4518 \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.

```
4519 \def\process@synonym#1{%
4520   \ifnum\last@language=\m@ne
4521     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4522   \else
4523     \expandafter\chardef\csname l@#1\endcsname\last@language
4524     \wlog{\string\l@#1=\string\language\the\last@language}%
4525     \expandafter\let\csname #1hyphenmins\endcsname\expandafter\endcsname
4526     \csname\language\hyphenmins\endcsname
4527     \let\bbl@elt\relax
4528     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4529   \fi}
```

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

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

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

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

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

Then we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group. When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

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

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

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

```

4530 \def\process@language#1#2#3{%
4531   \expandafter\addlanguage\csname l@#1\endcsname
4532   \expandafter\language\csname l@#1\endcsname
4533   \edef\language{#1}%
4534   \bbl@hook@everylanguage{#1}%
4535   % > luatex
4536   \bbl@get@enc#1::\@@@
4537   \begingroup
4538     \lefthyphenmin\m@ne
4539     \bbl@hook@loadpatterns{#2}%
4540     % > luatex
4541     \ifnum\lefthyphenmin=\m@ne
4542     \else
4543       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4544         \the\lefthyphenmin\the\righthyphenmin}%
4545       \fi
4546   \endgroup
4547   \def\bbl@tempa{#3}%
4548   \ifx\bbl@tempa\@empty\else
4549     \bbl@hook@loadexceptions{#3}%
4550     % > luatex
4551   \fi
4552   \let\bbl@elt\relax
4553   \edef\bbl@languages{%
4554     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4555   \ifnum\the\language=\z@
4556     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4557       \set@hyphenmins\tw@\thr@@\relax
4558     \else
4559       \expandafter\expandafter\expandafter\set@hyphenmins
4560       \csname #1hyphenmins\endcsname
4561     \fi
4562     \the\toks@
4563     \toks@{}%
4564   \fi}

```

`\bbl@get@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.

```

4565 \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.

```

4566 \def\bbl@hook@everylanguage#1{}
4567 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4568 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4569 \def\bbl@hook@loadkernel#1{%
4570   \def\addlanguage{\csname newlanguage\endcsname}%
4571   \def\adddialect##1##2{%

```

```

4572 \global\chardef##1##2\relax
4573 \wlog{\string##1 = a dialect from \string\language##2}}%
4574 \def\iflanguage##1{%
4575 \expandafter\ifx\csname l@##1\endcsname\relax
4576 \nolanner{##1}%
4577 \else
4578 \ifnum\csname l@##1\endcsname=\language
4579 \expandafter\expandafter\expandafter\@firstoftwo
4580 \else
4581 \expandafter\expandafter\expandafter\@secondoftwo
4582 \fi
4583 \fi}%
4584 \def\providehyphenmins##1##2{%
4585 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4586 \namedef{##1hyphenmins}{##2}%
4587 \fi}%
4588 \def\set@hyphenmins##1##2{%
4589 \lefthyphenmin##1\relax
4590 \righthyphenmin##2\relax}%
4591 \def\selectlanguage{%
4592 \errhelp{Selecting a language requires a package supporting it}%
4593 \errmessage{Not loaded}}%
4594 \let\foreignlanguage\selectlanguage
4595 \let\otherlanguage\selectlanguage
4596 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4597 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4598 \def\setlocale{%
4599 \errhelp{Find an armchair, sit down and wait}%
4600 \errmessage{(babel) Not yet available}}%
4601 \let\uselocale\setlocale
4602 \let\locale\setlocale
4603 \let\selectlocale\setlocale
4604 \let\localename\setlocale
4605 \let\textlocale\setlocale
4606 \let\textlanguage\setlocale
4607 \let\languagetext\setlocale}
4608 \begingroup
4609 \def\AddBabelHook#1#2{%
4610 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4611 \def\next{\toks1}%
4612 \else
4613 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4614 \fi
4615 \next}
4616 \ifx\directlua\@undefined
4617 \ifx\XeTeXinputencoding\@undefined\else
4618 \input xebabel.def
4619 \fi
4620 \else
4621 \input luababel.def
4622 \fi
4623 \openin1 = babel-\bbl@format.cfg
4624 \ifeof1
4625 \else
4626 \input babel-\bbl@format.cfg\relax
4627 \fi
4628 \closein1
4629 \endgroup
4630 \bbl@hook@loadkernel{switch.def}

```

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

```
4631 \openin1 = language.dat
```

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

about this.

```
4632 \def\language{english}%
4633 \ifeof1
4634   \message{I couldn't find the file language.dat,\space
4635           I will try the file hyphen.tex}
4636   \input hyphen.tex\relax
4637   \chardef\l@english\z@
4638 \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`.

```
4639 \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.

```
4640 \loop
4641   \endlinechar\m@ne
4642   \read1 to \bbl@line
4643   \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.

```
4644   \if T\ifeof1\fi T\relax
4645   \ifx\bbl@line\@empty\else
4646     \edef\bbl@line{\bbl@line\space\space\space}%
4647     \expandafter\process@line\bbl@line\relax
4648   \fi
4649 \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.

```
4650 \begingroup
4651   \def\bbl@elt#1#2#3#4{%
4652     \global\language=#2\relax
4653     \gdef\language{#1}%
4654     \def\bbl@elt##1##2##3##4{}}%
4655   \bbl@languages
4656 \endgroup
4657 \fi
4658 \closein1
```

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

```
4659 \if/\the\toks@\else
4660   \errhelp{language.dat loads no language, only synonyms}
4661   \errmessage{Orphan language synonym}
4662 \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.

```
4663 \let\bbl@line\@undefined
4664 \let\process@line\@undefined
4665 \let\process@synonym\@undefined
4666 \let\process@language\@undefined
4667 \let\bbl@get@enc\@undefined
4668 \let\bbl@hyph@enc\@undefined
4669 \let\bbl@tempa\@undefined
4670 \let\bbl@hook@loadkernel\@undefined
4671 \let\bbl@hook@everylanguage\@undefined
4672 \let\bbl@hook@loadpatterns\@undefined
```

```

4673 \let\bbl@hook@loadexceptions\@undefined
4674 \end{patterns}

```

Here the code for `initTeX` ends.

8 Font handling with fontspec

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` [misplaced].

```

4675 <<More package options>> ≡
4676 \chardef\bbl@bidimode\z@
4677 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4678 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4679 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4680 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4681 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4682 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4683 <</More package options>>

```

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`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to `babel`, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading (and mostly unuseful) message.

```

4684 <<Font selection>> ≡
4685 \bbl@trace{Font handling with fontspec}
4686 \ifx\ExplSyntaxOn\@undefined\else
4687   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4688     \in@{, #1, }{, no-script, language-not-exist, }%
4689     \ifin@ \else \bbl@tempfs@nx{#1}{#2}\fi}
4690   \def\bbl@fs@warn@nxx#1#2#3{%
4691     \in@{, #1, }{, no-script, language-not-exist, }%
4692     \ifin@ \else \bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4693   \def\bbl@loadfontspec{%
4694     \let\bbl@loadfontspec\relax
4695     \ifx\fontspec\@undefined
4696       \usepackage{fontspec}%
4697     \fi}%
4698 \fi
4699 \onlypreamble\babelfont
4700 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4701   \bbl@foreach{#1}{%
4702     \expandafter\ifx\csname date##1\endcsname\relax
4703       \IfFileExists{babel-##1.tex}%
4704         {\babelprovide{##1}}%
4705       {}%
4706     \fi}%
4707   \edef\bbl@tempa{#1}%
4708   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4709   \bbl@loadfontspec
4710   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4711   \bbl@bblfont}
4712 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4713   \bbl@ifunset{\bbl@tempb family}%
4714   {\bbl@providfam{\bbl@tempb}}%
4715   {}%
4716   % For the default font, just in case:
4717   \bbl@ifunset{\bbl@lsys\languagenam}{\bbl@provide@lsys{\languagenam}}{}%
4718   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4719   {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4720   \bbl@exp{%
4721     \let<\bbl@bbl@tempb dflt@\languagenam>\<\bbl@bbl@tempb dflt@>%

```

```

4722      \\\bbl@font@set<\bbl@tempb dflt@\\language>%
4723      <\bbl@tempb default><\bbl@tempb family>}}%
4724      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4725      \bbl@csarg\def{\bbl@tempb dflt@##1}{<#1>{#2}}}}%

```

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

```

4726 \def\bbl@providefam#1{%
4727   \bbl@exp{%
4728     \\\newcommand<#1default>{% Just define it
4729     \\\bbl@add@list\\bbl@font@fams{#1}%
4730     \\\DeclareRobustCommand<#1family>{%
4731       \\\not@math@alphabet<#1family>\relax
4732       % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4733       \\\fontfamily<#1default>%
4734       <{ifx>\\UseHooks\\@undefined<else>\\UseHook{#1family}<fi>%
4735       \\\selectfont}%
4736       \\\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.

```

4737 \def\bbl@nostdfont#1{%
4738   \bbl@ifunset{bbl@WFF@f@family}%
4739   {\bbl@csarg\gdef{WFF@f@family}}% Flag, to avoid dupl warns
4740   \bbl@infowarn{The current font is not a babel standard family:\\%
4741     #1%
4742     \fontname\font\\%
4743     There is nothing intrinsically wrong with this warning, and\\%
4744     you can ignore it altogether if you do not need these\\%
4745     families. But if they are used in the document, you should be\\%
4746     aware 'babel' will not set Script and Language for them, so\\%
4747     you may consider defining a new family with \string\babelfont.\\%
4748     See the manual for further details about \string\babelfont.\\%
4749     Reported}}
4750   }%
4751   \gdef\bbl@switchfont{%
4752     \bbl@ifunset{bbl@lsys@\\language}{\bbl@provide@lsys{\\language}}}%
4753     \bbl@exp{% eg Arabic -> arabic
4754     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}%
4755     \bbl@foreach\bbl@font@fams{%
4756       \bbl@ifunset{bbl@##1dflt@\\language}% (1) language?
4757       {\bbl@ifunset{bbl@##1dflt*\\bbl@tempa}% (2) from script?
4758       {\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
4759       }% 123=F - nothing!
4760       {\bbl@exp{% 3=T - from generic
4761         \global\let<bbl@##1dflt@\\language>%
4762         <bbl@##1dflt@>}}}%
4763       {\bbl@exp{% 2=T - from script
4764         \global\let<bbl@##1dflt@\\language>%
4765         <bbl@##1dflt*\\bbl@tempa>}}}%
4766       }% 1=T - language, already defined
4767     \def\bbl@tempa{\bbl@nostdfont}}% TODO. Don't use \bbl@tempa
4768     \bbl@foreach\bbl@font@fams{% don't gather with prev for
4769       \bbl@ifunset{bbl@##1dflt@\\language}%
4770       {\bbl@cs{famrst@##1}%
4771       \global\bbl@csarg\let{famrst@##1}\relax}%
4772       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4773       \\\bbl@add\\originalTeX{
4774       \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4775       <##1default><##1family>{##1}%
4776       \\\bbl@font@set<bbl@##1dflt@\\language>% the main part!
4777       <##1default><##1family>}}}%
4778     \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`.

```

4779 \ifx\family\@undefined\else % if latex
4780 \ifcase\bbbl@engine % if pdftex
4781 \let\bbbl@cckstdfont\relax
4782 \else
4783 \def\bbbl@cckstdfont{%
4784 \begingroup
4785 \global\let\bbbl@cckstdfont\relax
4786 \let\bbbl@tempa\empty
4787 \bbbl@foreach\bbbl@font@fams{%
4788 \bbbl@ifunset{\bbbl@##1dflt@}%
4789 {\@nameuse{##1family}}%
4790 \bbbl@csarg\gdef{WFF@family}}% Flag
4791 \bbbl@exp{\bbbl@add\bbbl@tempa{* \<##1family>= \family\\}%
4792 \space\space\fontname\font\\}%
4793 \bbbl@csarg\xdef{##1dflt@}{family}%
4794 \expandafter\xdef\csname ##1default\endcsname{family}%
4795 {}}%
4796 \ifx\bbbl@tempa\empty\else
4797 \bbbl@infowarn{The following font families will use the default\\%
4798 settings for all or some languages:\\%
4799 \bbbl@tempa
4800 There is nothing intrinsically wrong with it, but\\%
4801 'babel' will no set Script and Language, which could\\%
4802 be relevant in some languages. If your document uses\\%
4803 these families, consider redefining them with \string\babelfont.\\%
4804 Reported}%
4805 \fi
4806 \endgroup}
4807 \fi
4808 \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 `\bbbl@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*`).

```

4809 \def\bbbl@font@set#1#2#3{% eg \bbbl@rmdflt@lang \rmdefault \rmfamily
4810 \bbbl@xin@{<>}{#1}%
4811 \ifin@
4812 \bbbl@exp{\bbbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
4813 \fi
4814 \bbbl@exp{% 'Unprotected' macros return prev values
4815 \def\#2{#1}% eg, \rmdefault{\bbbl@rmdflt@lang}
4816 \bbbl@ifsamestring{#2}{family}%
4817 {\#3%
4818 \bbbl@ifsamestring{family}{bfdefault}{\bfseries}}%
4819 \let\bbbl@tempa\relax%
4820 {}%
4821 % TODO - next should be global?, but even local does its job. I'm
4822 % still not sure -- must investigate:
4823 \def\bbbl@fontspec@set#1#2#3#4{% eg \bbbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4824 \let\bbbl@tempe\bbbl@mapselect
4825 \edef\bbbl@tempb{\bbbl@stripslash#4}% Catcodes hack (better pass it).
4826 \bbbl@exp{\bbbl@replace\bbbl@tempb{\bbbl@stripslashfamily/}}%
4827 \let\bbbl@mapselect\relax
4828 \let\bbbl@tempa\tempfam#4% eg, '\rmfamily', to be restored below

```

```

4829 \let#4@empty % Make sure \renewfontfamily is valid
4830 \bbl@exp{%
4831 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4832 <\keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbl@ccl{sname}}}%
4833 {\\\newfontscript{\bbl@ccl{sname}}{\bbl@ccl{sotf}}}%
4834 <\keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbl@ccl{lname}}}%
4835 {\\\newfontlanguage{\bbl@ccl{lname}}{\bbl@ccl{lotf}}}%
4836 \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4837 \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4838 \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4839 \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4840 \\renewfontfamily\\#4%
4841 [\bbl@ccl{lsys},%
4842 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4843 #2]}{#3}% ie \bbl@exp{..}{#3}
4844 \bbl@exp{%
4845 \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4846 \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4847 \beginngroup
4848 #4%
4849 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4850 \endgroup % TODO. Find better tests:
4851 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4852 {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4853 \ifin@
4854 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4855 \fi
4856 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4857 {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4858 \ifin@
4859 \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4860 \fi
4861 \let#4\bbl@temp@fam
4862 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4863 \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.

```

4864 \def\bbl@font@rst#1#2#3#4{%
4865 \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4866 \def\bbl@font@fams{rm,sf,tt}
4867 <</Font selection>>

```

9 Hooks for XeTeX and LuaTeX

9.1 XeTeX

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

```

4868 <<(*Footnote changes)>> ≡
4869 \bbl@trace{Bidi footnotes}
4870 \ifnum\bbl@bidimode>\z@ % Any bidi=
4871 \def\bbl@footnote#1#2#3{%
4872 \@@ifnextchar[%
4873 {\bbl@footnote@o{#1}{#2}{#3}}%
4874 {\bbl@footnote@x{#1}{#2}{#3}}}
4875 \long\def\bbl@footnote@x#1#2#3#4{%
4876 \bgroup
4877 \select@language@x{\bbl@main@language}%
4878 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%

```



```

4879 \egroup}
4880 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4881 \bgroup
4882 \select@language@x{\bbl@main@language}%
4883 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4884 \egroup}
4885 \def\bbl@footnotetext#1#2#3{%
4886 \@ifnextchar[%
4887 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4888 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4889 \long\def\bbl@footnotetext@x#1#2#3#4{%
4890 \bgroup
4891 \select@language@x{\bbl@main@language}%
4892 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4893 \egroup}
4894 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4895 \bgroup
4896 \select@language@x{\bbl@main@language}%
4897 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4898 \egroup}
4899 \def\BabelFootnote#1#2#3#4{%
4900 \ifx\bbl@fn@footnote\@undefined
4901 \let\bbl@fn@footnote\footnote
4902 \fi
4903 \ifx\bbl@fn@footnotetext\@undefined
4904 \let\bbl@fn@footnotetext\footnotetext
4905 \fi
4906 \bbl@ifblank{#2}%
4907 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4908 \@namedef{\bbl@stripslash#1text}%
4909 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4910 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4911 \@namedef{\bbl@stripslash#1text}%
4912 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4913 \fi
4914 <\/Footnote changes>

```

Now, the code.

```

4915 <*\xetex>
4916 \def\BabelStringsDefault{unicode}
4917 \let\xebbl@stop\relax
4918 \AddBabelHook{xetex}{encodedcommands}{%
4919 \def\bbl@tempa{#1}%
4920 \ifx\bbl@tempa\@empty
4921 \XeTeXinputencoding"bytes"%
4922 \else
4923 \XeTeXinputencoding"#1"%
4924 \fi
4925 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4926 \AddBabelHook{xetex}{stopcommands}{%
4927 \xebbl@stop
4928 \let\xebbl@stop\relax}
4929 \def\bbl@intraspace#1 #2 #3\@{
4930 \bbl@csarg\gdef{xeisp@\languagename}%
4931 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4932 \def\bbl@intrapenalty#1\@{
4933 \bbl@csarg\gdef{xeipn@\languagename}%
4934 {\XeTeXlinebreakpenalty #1\relax}}
4935 \def\bbl@provide@intraspace{%
4936 \bbl@xin@{/s}{\bbl@c{l}{lnbrk}}}%
4937 \ifin@else\bbl@xin@{/c}{\bbl@c{l}{lnbrk}}\fi
4938 \ifin@
4939 \bbl@ifunset{bbl@intsp@\languagename}{%

```

```

4940 {\expandafter\ifx\csname bbl@intsp@\language\endcsname\@empty\else
4941 \ifx\bbl@KVP@intraspace\@nnil
4942 \bbl@exp{%
4943 \\\bbl@intraspace\bbl@cl{intsp}\@}%
4944 \fi
4945 \ifx\bbl@KVP@intrapenalty\@nnil
4946 \bbl@intrapenalty0\@
4947 \fi
4948 \fi
4949 \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4950 \expandafter\bbl@intraspace\bbl@KVP@intraspace\@
4951 \fi
4952 \ifx\bbl@KVP@intrapenalty\@nnil\else
4953 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
4954 \fi
4955 \bbl@exp{%
4956 % TODO. Execute only once (but redundant):
4957 \\\bbl@add\<extras\language>{%
4958 \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4959 \<bbl@xeisp@\language>%
4960 \<bbl@xeipn@\language>}%
4961 \\\bbl@toglobal\<extras\language>%
4962 \\\bbl@add\<noextras\language>{%
4963 \XeTeXlinebreaklocale ""}%
4964 \\\bbl@toglobal\<noextras\language>}%
4965 \ifx\bbl@ispace\@undefined
4966 \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4967 \ifx\AtBeginDocument\@notprerr
4968 \expandafter\@secondoftwo % to execute right now
4969 \fi
4970 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4971 \fi}%
4972 \fi}
4973 \ifx\DisableBabelHook\@undefined\endinput\fi
4974 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4975 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckcstdfont}
4976 \DisableBabelHook{babel-fontspec}
4977 <<Font selection>>
4978 \def\bbl@provide@extra#1{}

```

10 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.

```

4979 \ifnum\Xe@alloc@intercharclass<\thr@@
4980 \Xe@alloc@intercharclass\thr@@
4981 \fi
4982 \chardef\bbl@xe@class@default=\z@
4983 \chardef\bbl@xe@class@cjkideogram=\@ne
4984 \chardef\bbl@xe@class@cjkleftpunctuation=\tw@
4985 \chardef\bbl@xe@class@cjkrightpunctuation=\thr@@
4986 \chardef\bbl@xe@class@boundary=4095
4987 \chardef\bbl@xe@class@ignore=4096

```

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

```

4988 \AddBabelHook{babel-interchar}{beforeextras}{%
4989 \@nameuse{\bbl@xe@chars@\language}}
4990 \DisableBabelHook{babel-interchar}
4991 \protected\def\bbl@charclass#1{%

```

```

4992 \ifnum\count@<\z@
4993   \count@-\count@
4994   \loop
4995     \bbl@exp{%
4996       \\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4997       \XeTeXcharclass\count@ \bbl@tempc
4998       \ifnum\count@<`#1\relax
4999         \advance\count@ \@ne
5000       \repeat
5001   \else
5002     \babel@savevariable{\XeTeXcharclass`#1}%
5003     \XeTeXcharclass`#1 \bbl@tempc
5004   \fi
5005   \count@`#1\relax}

```

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

```

5006 \newcommand\IfBabelIntercharT[1]{%
5007   \let\bbl@tempa\@gobble      % Assume to ignore
5008   \edef\bbl@tempb{\zap@space#1 \@empty}%
5009   \ifx\bbl@KVP@interchar\@nnil\else
5010     \bbl@replace\bbl@KVP@interchar{ }{,}%
5011     \bbl@foreach\bbl@tempb{%
5012       \bbl@xin@{,##1,}{, \bbl@KVP@interchar,}%
5013       \ifin@
5014         \let\bbl@tempa\@firstofone
5015       \fi}%
5016   \fi
5017   \bbl@tempa}
5018 \newcommand\babelcharclass[3]{%
5019   \EnableBabelHook{babel-interchar}%
5020   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
5021   \def\bbl@tempb##1{%
5022     \ifx##1\@empty\else
5023       \ifx##1-
5024         \bbl@upto
5025       \else
5026         \bbl@charclass{%
5027           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
5028         \fi
5029         \expandafter\bbl@tempb
5030       \fi}%
5031   \bbl@ifunset{\bbl@xechars@#1}%
5032   {\toks@{%
5033     \babel@savevariable\XeTeXinterchartokenstate
5034     \XeTeXinterchartokenstate\@ne
5035   }}%
5036   {\toks@\expandafter\expandafter\expandafter{%
5037     \csname bbl@xechars@#1\endcsname}}%
5038   \bbl@csarg\edef{xchars@#1}{%
5039     \the\toks@
5040     \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
5041     \bbl@tempb#3\@empty}}
5042 \protected\def\bbl@usingxeclass#1{\count@\z@ \let\bbl@tempc#1}
5043 \protected\def\bbl@upto{%
5044   \ifnum\count@>\z@
5045     \advance\count@\@ne
5046     \count@-\count@
5047   \else\ifnum\count@=\z@

```

```

5048 \bbl@charclass{-}%
5049 \else
5050 \bbl@error{double-hyphens-class}{-}{-}%
5051 \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>@<lang>.

5052 \newcommand\babelinterchar[5][{}]{%
5053 \let\bbl@kv@label\@empty
5054 \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
5055 \@namedef{\zap@space\bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
5056 {\ifnum\language=\@nohyphenation
5057 \expandafter\@gobble
5058 \else
5059 \expandafter\@firstofone
5060 \fi
5061 {#5}}%
5062 \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
5063 \bbl@exp{\bbl@for{\bbl@tempa{\zap@space#3 \@empty}}{%
5064 \bbl@exp{\bbl@for{\bbl@tempb{\zap@space#4 \@empty}}{%
5065 \XeTeXinterchartoks
5066 \nameuse{bbl@xeclass@\bbl@tempa @#2}{#2}} %
5067 \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{#2}} %
5068 \nameuse{bbl@xeclass@\bbl@tempb @#2}{#2}} %
5069 \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{#2}} %
5070 = \expandafter{%
5071 \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
5072 \csname\zap@space\bbl@xeinter@\bbl@kv@label
5073 @#3@#4@#2 \@empty\endcsname}}}}
5074 \DeclareRobustCommand\enablelocaleinterchar[1]{%
5075 \bbl@ifunset{bbl@ic@#1@\language}%
5076 {\bbl@error{unknown-interchar}{#1}{-}}%
5077 {\bbl@csarg\let{ic@#1@\language}\@firstofone}}
5078 \DeclareRobustCommand\disablelocaleinterchar[1]{%
5079 \bbl@ifunset{bbl@ic@#1@\language}%
5080 {\bbl@error{unknown-interchar-b}{#1}{-}}%
5081 {\bbl@csarg\let{ic@#1@\language}\@gobble}}
5082 \let\@xetex

```

10.1 Layout

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

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

```

5083 (*xetex | textet)
5084 \providecommand\bbl@provide@intraspace{}
5085 \bbl@trace{Redefinitions for bidi layout}
5086 \def\bbl@sspre@caption{%
5087 \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
5088 \ifx\bbl@opt@layout\@nnil\else % if layout=..
5089 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
5090 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
5091 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
5092 \def\@hangfrom#1{%
5093 \setbox\@tempboxa\hbox{#1}}%
5094 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
5095 \noindent\box\@tempboxa}
5096 \def\raggedright{%
5097 \let\@centercr

```

```

5098 \bbl@startskip\z@skip
5099 \@rightskip\@flushglue
5100 \bbl@endskip\@rightskip
5101 \parindent\z@
5102 \parfillskip\bbl@startskip}
5103 \def\raggedleft{%
5104 \let\\\@centercr
5105 \bbl@startskip\@flushglue
5106 \bbl@endskip\z@skip
5107 \parindent\z@
5108 \parfillskip\bbl@endskip}
5109 \fi
5110 \IfBabelLayout{lists}
5111 {\bbl@sreplace\list
5112 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
5113 \def\bbl@listleftmargin{%
5114 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
5115 \ifcase\bbl@engine
5116 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
5117 \def\p@enumiii{\p@enumii}\theenumii{}\fi
5118 \fi
5119 \bbl@sreplace\@verbatim
5120 {\leftskip\@totalleftmargin}%
5121 {\bbl@startskip\textwidth
5122 \advance\bbl@startskip-\linewidth}%
5123 \bbl@sreplace\@verbatim
5124 {\rightskip\z@skip}%
5125 {\bbl@endskip\z@skip}}%
5126 {}
5127 \IfBabelLayout{contents}
5128 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
5129 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
5130 {}
5131 \IfBabelLayout{columns}
5132 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
5133 \def\bbl@outputbox#1{%
5134 \hb@xt@\textwidth{%
5135 \hskip\columnwidth
5136 \hfil
5137 {\normalcolor\vrule \@width\columnseprule}%
5138 \hfil
5139 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
5140 \hskip-\textwidth
5141 \hb@xt@\columnwidth{\box\@outputbox \hss}%
5142 \hskip\columnsep
5143 \hskip\columnwidth}}}%
5144 {}
5145 <<Footnote changes>>
5146 \IfBabelLayout{footnotes}%
5147 {\BabelFootnote\footnote\languagename{}\fi}%
5148 \BabelFootnote\localfootnote\languagename{}\fi}%
5149 \BabelFootnote\mainfootnote{}\fi}}
5150 {}

```

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

```

5151 \IfBabelLayout{counters*}%
5152 {\bbl@add\bbl@opt@layout{.counters.}%
5153 \AddToHook{shipout/before}{%
5154 \let\bbl@tempa\babelsublr
5155 \let\babelsublr\@firstofone
5156 \let\bbl@save@thepage\thepage
5157 \protected@edef\thepage{\thepage}%

```

```

5158 \let\babelsublr\bbl@tempa}%
5159 \AddToHook{shipout/after}{%
5160 \let\thepage\bbl@save@thepage}}{}
5161 \IfBabelLayout{counters}%
5162 {\let\bbl@latinarabic=@arabic
5163 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5164 \let\bbl@asciroman=@roman
5165 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5166 \let\bbl@asciiRoman=@Roman
5167 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
5168 \fi % end if layout
5169 </xetex | texxet>

```

10.2 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).

```

5170 <*texxet>
5171 \def\bbl@provide@extra#1{%
5172 % == auto-select encoding ==
5173 \ifx\bbl@encoding@select@off\@empty\else
5174 \bbl@ifunset{\bbl@encoding@#1}%
5175 {\def\@elt##1{,##1},%
5176 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5177 \count@\z@
5178 \bbl@foreach\bbl@tempe{%
5179 \def\bbl@tempd{##1}% Save last declared
5180 \advance\count@\@ne}%
5181 \ifnum\count@>\@ne % (1)
5182 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5183 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5184 \bbl@replace\bbl@tempa{ },}%
5185 \global\bbl@csarg\let{encoding@#1}\@empty
5186 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5187 \ifin@else % if main encoding included in ini, do nothing
5188 \let\bbl@tempb\relax
5189 \bbl@foreach\bbl@tempa{%
5190 \ifx\bbl@tempb\relax
5191 \bbl@xin@{,##1,}{,\bbl@tempe,}%
5192 \ifin@def\bbl@tempb{##1}\fi
5193 \fi}%
5194 \ifx\bbl@tempb\relax\else
5195 \bbl@exp{%
5196 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>}%
5197 \gdef\<\bbl@encoding@#1>{%
5198 \\\babel@save\\f@encoding
5199 \\\bbl@add\\originalTeX{\\selectfont}%
5200 \\\fontencoding{\bbl@tempb}%
5201 \\\selectfont}}%
5202 \fi
5203 \fi
5204 \fi}%
5205 {}%
5206 \fi}
5207 </texxet>

```

10.3 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` (eg, `\babelpatterns`).

```

5208 (*luatex)
5209 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5210 \bbl@trace{Read language.dat}
5211 \ifx\bbl@readstream\undefined
5212   \csname newread\endcsname\bbl@readstream
5213 \fi
5214 \begingroup
5215   \toks@{}
5216   \count@ \z@ % 0=start, 1=0th, 2=normal
5217   \def\bbl@process@line#1#2 #3 #4 {%
5218     \ifx=#1%
5219       \bbl@process@synonym{#2}%
5220     \else
5221       \bbl@process@language{#1#2}{#3}{#4}%
5222     \fi
5223     \ignorespaces}
5224   \def\bbl@manylang{%
5225     \ifnum\bbl@last>\@ne
5226       \bbl@info{Non-standard hyphenation setup}%
5227     \fi
5228     \let\bbl@manylang\relax}
5229   \def\bbl@process@language#1#2#3{%
5230     \ifcase\count@
5231       \@ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
5232     \or
5233       \count@\tw@
5234     \fi
5235     \ifnum\count@=\tw@
5236       \expandafter\addlanguage\csname l@#1\endcsname
5237       \language\allocationnumber
5238       \chardef\bbl@last\allocationnumber
5239       \bbl@manylang
5240       \let\bbl@elt\relax
5241       \xdef\bbl@languages{%
5242         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
```

```

5243 \fi
5244 \the\toks@
5245 \toks@{}}
5246 \def\bbl@process@synonym@aux#1#2{%
5247 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5248 \let\bbl@elt\relax
5249 \xdef\bbl@languages{%
5250 \bbl@languages\bbl@elt{#1}{#2}{}}}%
5251 \def\bbl@process@synonym#1{%
5252 \ifcase\count@
5253 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5254 \or
5255 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5256 \else
5257 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5258 \fi}
5259 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
5260 \chardef\l@english\z@
5261 \chardef\l@USenglish\z@
5262 \chardef\bbl@last\z@
5263 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5264 \gdef\bbl@languages{%
5265 \bbl@elt{english}{0}{hyphen.tex}}%
5266 \bbl@elt{USenglish}{0}{}}
5267 \else
5268 \global\let\bbl@languages@format\bbl@languages
5269 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5270 \ifnum#2>\z@
5271 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5272 \fi}%
5273 \xdef\bbl@languages{\bbl@languages}%
5274 \fi
5275 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}} % Define flags
5276 \bbl@languages
5277 \openin\bbl@readstream=language.dat
5278 \ifeof\bbl@readstream
5279 \bbl@warning{I couldn't find language.dat. No additional\\%
5280 patterns loaded. Reported}%
5281 \else
5282 \loop
5283 \endlinechar\m@ne
5284 \read\bbl@readstream to \bbl@line
5285 \endlinechar\^M
5286 \if T\ifeof\bbl@readstream F\fi T\relax
5287 \ifx\bbl@line\@empty\else
5288 \edef\bbl@line{\bbl@line\space\space\space}%
5289 \expandafter\bbl@process@line\bbl@line\relax
5290 \fi
5291 \repeat
5292 \fi
5293 \closein\bbl@readstream
5294 \endgroup
5295 \bbl@trace{Macros for reading patterns files}
5296 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5297 \ifx\babelcatcodetablenum\@undefined
5298 \ifx\newcatcodetable\@undefined
5299 \def\babelcatcodetablenum{5211}
5300 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5301 \else
5302 \newcatcodetable\babelcatcodetablenum
5303 \newcatcodetable\bbl@pattcodes
5304 \fi
5305 \else

```



```

5306 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5307 \fi
5308 \def\bbl@luapatterns#1#2{%
5309 \bbl@get@enc#1::\@@@
5310 \setbox\z@\hbox\bgroup
5311 \begingroup
5312 \savecatcodetable\babelcatcodetablenum\relax
5313 \initcatcodetable\bbl@pattcodes\relax
5314 \catcodetable\bbl@pattcodes\relax
5315 \catcode\#=6 \catcode\$_=3 \catcode\&=4 \catcode\^=7
5316 \catcode\_ =8 \catcode\{=1 \catcode\}=2 \catcode\~=13
5317 \catcode\@=11 \catcode\^^I=10 \catcode\^^J=12
5318 \catcode\<=12 \catcode\>=12 \catcode\*=12 \catcode\.=12
5319 \catcode\-=12 \catcode\/=12 \catcode\[=12 \catcode\]=12
5320 \catcode\`=12 \catcode\'=12 \catcode\"=12
5321 \input #1\relax
5322 \catcodetable\babelcatcodetablenum\relax
5323 \endgroup
5324 \def\bbl@tempa{#2}%
5325 \ifx\bbl@tempa\@empty\else
5326 \input #2\relax
5327 \fi
5328 \egroup}%
5329 \def\bbl@patterns@lua#1{%
5330 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5331 \csname l@#1\endcsname
5332 \edef\bbl@tempa{#1}%
5333 \else
5334 \csname l@#1:f@encoding\endcsname
5335 \edef\bbl@tempa{#1:f@encoding}%
5336 \fi\relax
5337 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5338 \@ifundefined{bbl@hyphendata@the\language}%
5339 { \def\bbl@elt##1##2##3##4{%
5340 \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5341 \def\bbl@tempb{##3}%
5342 \ifx\bbl@tempb\@empty\else % if not a synonymous
5343 \def\bbl@tempc{{##3}{##4}}%
5344 \fi
5345 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5346 \fi}%
5347 \bbl@languages
5348 \@ifundefined{bbl@hyphendata@the\language}%
5349 {\bbl@info{No hyphenation patterns were set for\\%
5350 language '\bbl@tempa'. Reported}}%
5351 {\expandafter\expandafter\expandafter\bbl@luapatterns
5352 \csname bbl@hyphendata@the\language\endcsname}}}%
5353 \endinput\fi
5354 % Here ends \ifx\AddBabelHook\@undefined
5355 % A few lines are only read by hyphen.cfg
5356 \ifx\DisableBabelHook\@undefined
5357 \AddBabelHook{luatex}{everylanguage}{%
5358 \def\process@language##1##2##3{%
5359 \def\process@line####1####2 ####3 ####4 {}}%
5360 \AddBabelHook{luatex}{loadpatterns}{%
5361 \input #1\relax
5362 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5363 {{#1}}}%
5364 \AddBabelHook{luatex}{loadexceptions}{%
5365 \input #1\relax
5366 \def\bbl@tempb##1##2{{##1}{##2}}%
5367 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5368 {\expandafter\expandafter\expandafter\bbl@tempb

```

```

5369         \csname bbl@hyphendata@the\language\endcsname}}
5370 \endinput\fi
5371 % Here stops reading code for hyphen.cfg
5372 % The following is read the 2nd time it's loaded
5373 \begingroup % TODO - to a lua file
5374 \catcode`\%=12
5375 \catcode`\'=12
5376 \catcode`\ "=12
5377 \catcode`\:=12
5378 \directlua{
5379   Babel = Babel or {}
5380   function Babel.bytes(line)
5381     return line:gsub(".",
5382       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5383   end
5384   function Babel.begin_process_input()
5385     if luatexbase and luatexbase.add_to_callback then
5386       luatexbase.add_to_callback('process_input_buffer',
5387         Babel.bytes, 'Babel.bytes')
5388     else
5389       Babel.callback = callback.find('process_input_buffer')
5390       callback.register('process_input_buffer', Babel.bytes)
5391     end
5392   end
5393   function Babel.end_process_input ()
5394     if luatexbase and luatexbase.remove_from_callback then
5395       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5396     else
5397       callback.register('process_input_buffer', Babel.callback)
5398     end
5399   end
5400   function Babel.addpatterns(pp, lg)
5401     local lg = lang.new(lg)
5402     local pats = lang.patterns(lg) or ''
5403     lang.clear_patterns(lg)
5404     for p in pp:gmatch('[^%s]+') do
5405       ss = ''
5406       for i in string.utfcharacters(p:gsub('%d', '')) do
5407         ss = ss .. '%d?' .. i
5408       end
5409       ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5410       ss = ss:gsub('%.%d%?$', '%%.')
5411       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5412       if n == 0 then
5413         tex.sprint(
5414           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5415           .. p .. [[{}]])
5416         pats = pats .. ' ' .. p
5417       else
5418         tex.sprint(
5419           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5420           .. p .. [[{}]])
5421       end
5422     end
5423     lang.patterns(lg, pats)
5424   end
5425   Babel.characters = Babel.characters or {}
5426   Babel.ranges = Babel.ranges or {}
5427   function Babel.hlist_has_bidi(head)
5428     local has_bidi = false
5429     local ranges = Babel.ranges
5430     for item in node.traverse(head) do
5431       if item.id == node.id'glyph' then

```

```

5432     local itemchar = item.char
5433     local chardata = Babel.characters[itemchar]
5434     local dir = chardata and chardata.d or nil
5435     if not dir then
5436         for nn, et in ipairs(ranges) do
5437             if itemchar < et[1] then
5438                 break
5439             elseif itemchar <= et[2] then
5440                 dir = et[3]
5441                 break
5442             end
5443         end
5444     end
5445     if dir and (dir == 'al' or dir == 'r') then
5446         has_bidi = true
5447     end
5448 end
5449 end
5450 return has_bidi
5451 end
5452 function Babel.set_chranges_b (script, chrng)
5453     if chrng == '' then return end
5454     texio.write('Replacing ' .. script .. ' script ranges')
5455     Babel.script_blocks[script] = {}
5456     for s, e in string.gmatch(chrng..' ', '(.-%.(-(.)%s)') do
5457         table.insert(
5458             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5459     end
5460 end
5461 function Babel.discard_sublr(str)
5462     if str:find( [[\string\indexentry]] ) and
5463        str:find( [[\string\babelsublr]] ) then
5464         str = str:gsub( [[\string\babelsublr%*(%b{})]],
5465                        function(m) return m:sub(2,-2) end )
5466     end
5467     return str
5468 end
5469 }
5470 \endgroup
5471 \ifx\newattribute\undefined\else % Test for plain
5472 \newattribute\bbl@attr@locale
5473 \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5474 \AddBabelHook{luatex}{beforeextras}{%
5475     \setattribute\bbl@attr@locale\localeid}
5476 \fi
5477 \def\BabelStringsDefault{unicode}
5478 \let\luabbl@stop\relax
5479 \AddBabelHook{luatex}{encodedcommands}{%
5480     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5481     \ifx\bbl@tempa\bbl@tempb\else
5482         \directlua{Babel.begin_process_input()}%
5483         \def\luabbl@stop{%
5484             \directlua{Babel.end_process_input()}}%
5485     \fi}%
5486 \AddBabelHook{luatex}{stopcommands}{%
5487     \luabbl@stop
5488     \let\luabbl@stop\relax}
5489 \AddBabelHook{luatex}{patterns}{%
5490     \ifundefined{bbl@hyphendata@\the\language}%
5491     {\def\bbl@elt##1##2##3##4{%
5492         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5493         \def\bbl@tempb{##3}%
5494         \ifx\bbl@tempb\empty\else % if not a synonymous

```

```

5495         \def\bbl@tempc{{##3}{##4}}%
5496     \fi
5497     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5498 \fi}%
5499 \bbl@languages
5500 \@ifundefined{bbl@hyphendata@the\language}%
5501     {\bbl@info{No hyphenation patterns were set for\\%
5502         language '#2'. Reported}}%
5503     {\expandafter\expandafter\expandafter\bbl@luapatterns
5504         \csname bbl@hyphendata@the\language\endcsname}}}%
5505 \@ifundefined{bbl@patterns@}{}%
5506 \begingroup
5507     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5508 \ifin\else
5509     \ifx\bbl@patterns@\empty\else
5510         \directlua{ Babel.addpatterns(
5511             [[\bbl@patterns@]], \number\language) }%
5512     \fi
5513     \@ifundefined{bbl@patterns@#1}%
5514     \@empty
5515     {\directlua{ Babel.addpatterns(
5516         [[\space\csname bbl@patterns@#1\endcsname]],
5517         \number\language) }}%
5518     \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5519 \fi
5520 \endgroup}%
5521 \bbl@exp{%
5522     \bbl@ifunset{bbl@prehc@\language\name}{}%
5523     {\bbl@ifblank{\bbl@cs{prehc@\language\name}}{}}%
5524     {\prehyphenchar=\bbl@c{prehc}\relax}}}%

```

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

```

5525 \@onlypreamble\babelpatterns
5526 \AtEndOfPackage{%
5527     \newcommand\babelpatterns[2][\empty]{%
5528         \ifx\bbl@patterns@\relax
5529             \let\bbl@patterns@\empty
5530         \fi
5531         \ifx\bbl@pttnlist@\empty\else
5532             \bbl@warning{%
5533                 You must not intermingle \string\selectlanguage\space and\\%
5534                 \string\babelpatterns\space or some patterns will not\\%
5535                 be taken into account. Reported}%
5536             \fi
5537             \ifx\@empty#1%
5538                 \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5539             \else
5540                 \edef\bbl@tempb{\zap@space#1 \@empty}%
5541                 \bbl@for\bbl@tempa\bbl@tempb{%
5542                     \bbl@fixname\bbl@tempa
5543                     \bbl@iflanguage\bbl@tempa{%
5544                         \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5545                             \@ifundefined{bbl@patterns@\bbl@tempa}%
5546                             \@empty
5547                             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5548                             #2}}}%
5549                 \fi}}

```

10.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.

Replace regular (ie, 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.

```

5550% TODO - to a lua file
5551\directlua{
5552  Babel = Babel or {}
5553  Babel.linebreaking = Babel.linebreaking or {}
5554  Babel.linebreaking.before = {}
5555  Babel.linebreaking.after = {}
5556  Babel.locale = {} % Free to use, indexed by \localeid
5557  function Babel.linebreaking.add_before(func, pos)
5558    tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname]])
5559    if pos == nil then
5560      table.insert(Babel.linebreaking.before, func)
5561    else
5562      table.insert(Babel.linebreaking.before, pos, func)
5563    end
5564  end
5565  function Babel.linebreaking.add_after(func)
5566    tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname]])
5567    table.insert(Babel.linebreaking.after, func)
5568  end
5569}
5570\def\bbl@intraspace#1 #2 #3\@@{%
5571  \directlua{
5572    Babel = Babel or {}
5573    Babel.intraspaces = Babel.intraspaces or {}
5574    Babel.intraspaces['\csname bbl@sbcp@\language\endcsname'] = %
5575      {b = #1, p = #2, m = #3}
5576    Babel.locale_props[\the\localeid].intraspace = %
5577      {b = #1, p = #2, m = #3}
5578  }}
5579\def\bbl@intrapenalty#1\@@{%
5580  \directlua{
5581    Babel = Babel or {}
5582    Babel.intrapenalties = Babel.intrapenalties or {}
5583    Babel.intrapenalties['\csname bbl@sbcp@\language\endcsname'] = #1
5584    Babel.locale_props[\the\localeid].intrapenalty = #1
5585  }}
5586\beginingroup
5587\catcode`\%=12
5588\catcode`\^=14
5589\catcode`\'=12
5590\catcode`\-=12
5591\gdef\bbl@seaintraspace{^
5592  \let\bbl@seaintraspace\relax
5593  \directlua{
5594    Babel = Babel or {}
5595    Babel.sea_enabled = true
5596    Babel.sea_ranges = Babel.sea_ranges or {}
5597    function Babel.set_chranges (script, chrng)
5598      local c = 0
5599      for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5600        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5601        c = c + 1
5602      end
5603    end
5604    function Babel.sea_disc_to_space (head)
5605      local sea_ranges = Babel.sea_ranges
5606      local last_char = nil
5607      local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5608      for item in node.traverse(head) do
5609        local i = item.id

```

```

5610     if i == node.id'glyph' then
5611         last_char = item
5612     elseif i == 7 and item.subtype == 3 and last_char
5613         and last_char.char > 0x0C99 then
5614         quad = font.getfont(last_char.font).size
5615         for lg, rg in pairs(sea_ranges) do
5616             if last_char.char > rg[1] and last_char.char < rg[2] then
5617                 lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5618                 local intraspace = Babel.intraspaces[lg]
5619                 local intrapenalty = Babel.intrapenalties[lg]
5620                 local n
5621                 if intrapenalty ~= 0 then
5622                     n = node.new(14, 0) ^% penalty
5623                     n.penalty = intrapenalty
5624                     node.insert_before(head, item, n)
5625                 end
5626                 n = node.new(12, 13) ^% (glue, spaceskip)
5627                 node.setglue(n, intraspace.b * quad,
5628                     intraspace.p * quad,
5629                     intraspace.m * quad)
5630                 node.insert_before(head, item, n)
5631                 node.remove(head, item)
5632             end
5633         end
5634     end
5635 end
5636 end
5637 }^^
5638 \bbl@luahyphenate}

```

10.5 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.

```

5639 \catcode`\%=14
5640 \gdef\bbl@cjkintraspaces{%
5641     \let\bbl@cjkintraspaces\relax
5642     \directlua{
5643         Babel = Babel or {}
5644         require('babel-data-cjk.lua')
5645         Babel.cjk_enabled = true
5646         function Babel.cjk_linebreak(head)
5647             local GLYPH = node.id'glyph'
5648             local last_char = nil
5649             local quad = 655360 % 10 pt = 655360 = 10 * 65536
5650             local last_class = nil
5651             local last_lang = nil
5652
5653             for item in node.traverse(head) do
5654                 if item.id == GLYPH then
5655
5656                     local lang = item.lang
5657
5658                     local LOCALE = node.get_attribute(item,
5659                         Babel.attr_locale)
5660                     local props = Babel.locale_props[LOCALE]
5661
5662                     local class = Babel.cjk_class[item.char].c
5663

```

```

5664         if props.cjk_quotes and props.cjk_quotes[item.char] then
5665             class = props.cjk_quotes[item.char]
5666         end
5667
5668         if class == 'cp' then class = 'cl' end % ]] as CL
5669         if class == 'id' then class = 'I' end
5670
5671         local br = 0
5672         if class and last_class and Babel.cjk_breaks[last_class][class] then
5673             br = Babel.cjk_breaks[last_class][class]
5674         end
5675
5676         if br == 1 and props.linebreak == 'c' and
5677             lang ~= \the\l@nohyphenation\space and
5678             last_lang ~= \the\l@nohyphenation then
5679             local intrapenalty = props.intrapenalty
5680             if intrapenalty ~= 0 then
5681                 local n = node.new(14, 0)      % penalty
5682                 n.penalty = intrapenalty
5683                 node.insert_before(head, item, n)
5684             end
5685             local intraspace = props.intraspace
5686             local n = node.new(12, 13)        % (glue, spaceskip)
5687             node.setglue(n, intraspace.b * quad,
5688                           intraspace.p * quad,
5689                           intraspace.m * quad)
5690             node.insert_before(head, item, n)
5691         end
5692
5693         if font.getfont(item.font) then
5694             quad = font.getfont(item.font).size
5695         end
5696         last_class = class
5697         last_lang = lang
5698         else % if penalty, glue or anything else
5699             last_class = nil
5700         end
5701     end
5702     lang.hyphenate(head)
5703 end
5704 }%
5705 \bbl@luaohyphenate}
5706 \gdef\bbl@luaohyphenate{%
5707 \let\bbl@luaohyphenate\relax
5708 \directlua{
5709     luatexbase.add_to_callback('hyphenate',
5710     function (head, tail)
5711         if Babel.linebreaking.before then
5712             for k, func in ipairs(Babel.linebreaking.before) do
5713                 func(head)
5714             end
5715         end
5716         if Babel.cjk_enabled then
5717             Babel.cjk_linebreak(head)
5718         end
5719         lang.hyphenate(head)
5720         if Babel.linebreaking.after then
5721             for k, func in ipairs(Babel.linebreaking.after) do
5722                 func(head)
5723             end
5724         end
5725         if Babel.sea_enabled then
5726             Babel.sea_disc_to_space(head)

```

```

5727     end
5728   end,
5729   'Babel.hyphenate')
5730 }
5731 }
5732 \endgroup
5733 \def\bbl@provide@intraspace{%
5734   \bbl@ifunset{\bbl@intsp@{language}}{}%
5735   {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5736     \bbl@xin{/c}{/\bbl@cl{lnbrk}}}%
5737   \ifin@           % cjk
5738     \bbl@cjk@intraspace
5739     \directlua{
5740       Babel = Babel or {}
5741       Babel.locale_props = Babel.locale_props or {}
5742       Babel.locale_props[\the\localeid].linebreak = 'c'
5743     }%
5744     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5745     \ifx\bbl@KVP@intrapenalty\@nnil
5746       \bbl@intrapenalty0\@{}
5747     \fi
5748   \else           % sea
5749     \bbl@sea@intraspace
5750     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@{}%
5751     \directlua{
5752       Babel = Babel or {}
5753       Babel.sea_ranges = Babel.sea_ranges or {}
5754       Babel.set_chranges('\bbl@cl{sbcpr}',
5755                           '\bbl@cl{chrng}')
5756     }%
5757     \ifx\bbl@KVP@intrapenalty\@nnil
5758       \bbl@intrapenalty0\@{}
5759     \fi
5760   \fi
5761 \fi
5762 \ifx\bbl@KVP@intrapenalty\@nnil\else
5763   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@{}
5764 \fi}}

```

10.6 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-

```

5765 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5766 \def\bblar@chars{%
5767   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5768   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5769   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5770 \def\bblar@elongated{%
5771   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5772   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5773   0649,064A}
5774 \begin{group}
5775   \catcode`\_ = 11 \catcode`\:= 11
5776   \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5777 \end{group}
5778 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5779   \let\bbl@arabicjust\relax
5780   \newattribute\bblar@kashida
5781   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5782   \bblar@kashida=\z@
5783   \bbl@patchfont{\bbl@parsejalt}}%
5784   \directlua{

```



```

5785 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5786 Babel.arabic.elong_map[\the\localeid] = {}
5787 luatexbase.add_to_callback('post_linebreak_filter',
5788   Babel.arabic.justify, 'Babel.arabic.justify')
5789 luatexbase.add_to_callback('hpack_filter',
5790   Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5791 }%

```

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

```

5792 \def\bblar@fetchjalt#1#2#3#4{%
5793   \bbl@exp{\bbl@foreach{#1}}{%
5794     \bbl@ifunset\bblar@JE@##1}%
5795     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"##1#2}}%
5796     {\setbox\z@\hbox{\textdir TRT ^^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5797   \directlua{%
5798     local last = nil
5799     for item in node.traverse(tex.box[0].head) do
5800       if item.id == node.id'glyph' and item.char > 0x600 and
5801         not (item.char == 0x200D) then
5802         last = item
5803       end
5804     end
5805     Babel.arabic.#3['##1#4'] = last.char
5806   }}

```

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

```

5807 \gdef\bbl@parsejalt{%
5808   \ifx\addfontfeature\@undefined\else
5809     \bbl@xin@{/e}{/\bbl@c{l}{lnbrk}}%
5810     \ifin@
5811       \directlua{%
5812         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5813           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5814           tex.print([[string\curname\space bbl@parsejalti\endcurname]])
5815         end
5816       }%
5817     \fi
5818   \fi}
5819 \gdef\bbl@parsejalti{%
5820   \begingroup
5821     \let\bbl@parsejalt\relax % To avoid infinite loop
5822     \edef\bbl@tempb{\fontid\font}%
5823     \bblar@nofswarn
5824     \bblar@fetchjalt\bblar@elongated{}{from}}%
5825     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5826     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5827     \addfontfeature{RawFeature+=jalt}%
5828     % \@namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5829     \bblar@fetchjalt\bblar@elongated{}{dest}}%
5830     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5831     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5832     \directlua{%
5833       for k, v in pairs(Babel.arabic.from) do
5834         if Babel.arabic.dest[k] and
5835           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5836           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5837             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5838         end
5839       end
5840     }%
5841   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5842 \beginingroup
5843 \catcode`#=11
5844 \catcode`~=11
5845 \directlua{
5846
5847 Babel.arabic = Babel.arabic or {}
5848 Babel.arabic.from = {}
5849 Babel.arabic.dest = {}
5850 Babel.arabic.justify_factor = 0.95
5851 Babel.arabic.justify_enabled = true
5852 Babel.arabic.kashida_limit = -1
5853
5854 function Babel.arabic.justify(head)
5855   if not Babel.arabic.justify_enabled then return head end
5856   for line in node.traverse_id(node.id'hlist', head) do
5857     Babel.arabic.justify_hlist(head, line)
5858   end
5859   return head
5860 end
5861
5862 function Babel.arabic.justify_hbox(head, gc, size, pack)
5863   local has_inf = false
5864   if Babel.arabic.justify_enabled and pack == 'exactly' then
5865     for n in node.traverse_id(12, head) do
5866       if n.stretch_order > 0 then has_inf = true end
5867     end
5868     if not has_inf then
5869       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5870     end
5871   end
5872   return head
5873 end
5874
5875 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5876   local d, new
5877   local k_list, k_item, pos_inline
5878   local width, width_new, full, k_curr, wt_pos, goal, shift
5879   local subst_done = false
5880   local elong_map = Babel.arabic.elong_map
5881   local cnt
5882   local last_line
5883   local GLYPH = node.id'glyph'
5884   local KASHIDA = Babel.attr_kashida
5885   local LOCALE = Babel.attr_locale
5886
5887   if line == nil then
5888     line = {}
5889     line.glue_sign = 1
5890     line.glue_order = 0
5891     line.head = head
5892     line.shift = 0
5893     line.width = size
5894   end
5895
5896   % Exclude last line. todo. But-- it discards one-word lines, too!
5897   % ? Look for glue = 12:15
5898   if (line.glue_sign == 1 and line.glue_order == 0) then
5899     elongs = {} % Stores elongated candidates of each line
5900     k_list = {} % And all letters with kashida
5901     pos_inline = 0 % Not yet used
5902
5903     for n in node.traverse_id(GLYPH, line.head) do
5904       pos_inline = pos_inline + 1 % To find where it is. Not used.

```

```

5905
5906 % Elongated glyphs
5907 if elong_map then
5908     local locale = node.get_attribute(n, LOCALE)
5909     if elong_map[locale] and elong_map[locale][n.font] and
5910         elong_map[locale][n.font][n.char] then
5911         table.insert(elongs, {node = n, locale = locale} )
5912         node.set_attribute(n.prev, KASHIDA, 0)
5913     end
5914 end
5915
5916 % Tatwil
5917 if Babel.kashida_wts then
5918     local k_wt = node.get_attribute(n, KASHIDA)
5919     if k_wt > 0 then % todo. parameter for multi inserts
5920         table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5921     end
5922 end
5923
5924 end % of node.traverse_id
5925
5926 if #elongs == 0 and #k_list == 0 then goto next_line end
5927 full = line.width
5928 shift = line.shift
5929 goal = full * Babel.arabic.justify_factor % A bit crude
5930 width = node.dimensions(line.head) % The 'natural' width
5931
5932 % == Elongated ==
5933 % Original idea taken from 'chickenize'
5934 while (#elongs > 0 and width < goal) do
5935     subst_done = true
5936     local x = #elongs
5937     local curr = elongs[x].node
5938     local oldchar = curr.char
5939     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5940     width = node.dimensions(line.head) % Check if the line is too wide
5941     % Substitute back if the line would be too wide and break:
5942     if width > goal then
5943         curr.char = oldchar
5944         break
5945     end
5946     % If continue, pop the just substituted node from the list:
5947     table.remove(elongs, x)
5948 end
5949
5950 % == Tatwil ==
5951 if #k_list == 0 then goto next_line end
5952
5953 width = node.dimensions(line.head) % The 'natural' width
5954 k_curr = #k_list % Traverse backwards, from the end
5955 wt_pos = 1
5956
5957 while width < goal do
5958     subst_done = true
5959     k_item = k_list[k_curr].node
5960     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5961         d = node.copy(k_item)
5962         d.char = 0x0640
5963         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5964         d.xoffset = 0
5965         line.head, new = node.insert_after(line.head, k_item, d)
5966         width_new = node.dimensions(line.head)
5967         if width > goal or width == width_new then

```

```

5968         node.remove(line.head, new) % Better compute before
5969         break
5970     end
5971     if Babel.fix_diacr then
5972         Babel.fix_diacr(k_item.next)
5973     end
5974     width = width_new
5975 end
5976 if k_curr == 1 then
5977     k_curr = #k_list
5978     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5979 else
5980     k_curr = k_curr - 1
5981 end
5982 end
5983
5984 % Limit the number of tatweel by removing them. Not very efficient,
5985 % but it does the job in a quite predictable way.
5986 if Babel.arabic.kashida_limit > -1 then
5987     cnt = 0
5988     for n in node.traverse_id(GLYPH, line.head) do
5989         if n.char == 0x0640 then
5990             cnt = cnt + 1
5991             if cnt > Babel.arabic.kashida_limit then
5992                 node.remove(line.head, n)
5993             end
5994         else
5995             cnt = 0
5996         end
5997     end
5998 end
5999
6000 ::next_line::
6001
6002 % Must take into account marks and ins, see luatex manual.
6003 % Have to be executed only if there are changes. Investigate
6004 % what's going on exactly.
6005 if subst_done and not gc then
6006     d = node.hpack(line.head, full, 'exactly')
6007     d.shift = shift
6008     node.insert_before(head, line, d)
6009     node.remove(head, line)
6010 end
6011 end % if process line
6012 end
6013 }
6014 \endgroup
6015 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

10.7 Common stuff

```

6016 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
6017 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ccheckstdfonts}
6018 \DisableBabelHook{babel-fontspec}
6019 <<Font selection>>

```

10.8 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.

```

6020% TODO - to a lua file
6021\directlua{
6022Babel.script_blocks = {
6023  ['dflt'] = {},
6024  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
6025             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
6026  ['Armn'] = {{0x0530, 0x058F}},
6027  ['Beng'] = {{0x0980, 0x09FF}},
6028  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
6029  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
6030  ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
6031              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
6032  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
6033  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
6034              {0xAB00, 0xAB2F}},
6035  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
6036  % Don't follow strictly Unicode, which places some Coptic letters in
6037  % the 'Greek and Coptic' block
6038  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
6039  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
6040              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
6041              {0xF900, 0FAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
6042              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
6043              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
6044              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
6045  ['Hebr'] = {{0x0590, 0x05FF}},
6046  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
6047              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
6048  ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
6049  ['Knda'] = {{0x0C80, 0x0CFF}},
6050  ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
6051              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
6052              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
6053  ['Lao'] = {{0x0E80, 0x0EFF}},
6054  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
6055              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
6056              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
6057  ['Mahj'] = {{0x1150, 0x117F}},
6058  ['Mlym'] = {{0x0D00, 0x0D7F}},
6059  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
6060  ['Orya'] = {{0x0B00, 0x0B7F}},
6061  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x11E0, 0x11FF}},
6062  ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
6063  ['Taml'] = {{0x0B80, 0x0BFF}},
6064  ['Telu'] = {{0x0C00, 0x0C7F}},
6065  ['Tfng'] = {{0x2D30, 0x2D7F}},
6066  ['Thai'] = {{0x0E00, 0x0E7F}},
6067  ['Tibt'] = {{0x0F00, 0x0FFF}},
6068  ['Vaii'] = {{0xA500, 0xA63F}},
6069  ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
6070 }
6071
6072 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyril
6073 Babel.script_blocks.Hant = Babel.script_blocks.Hans
6074 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
6075
6076 function Babel.locale_map(head)
6077   if not Babel.locale_mapped then return head end
6078
6079   local LOCALE = Babel.attr_locale
6080   local GLYPH = node.id('glyph')

```

```

6081 local inmath = false
6082 local toloc_save
6083 for item in node.traverse(head) do
6084   local toloc
6085   if not inmath and item.id == GLYPH then
6086     % Optimization: build a table with the chars found
6087     if Babel.chr_to_loc[item.char] then
6088       toloc = Babel.chr_to_loc[item.char]
6089     else
6090       for lc, maps in pairs(Babel.loc_to_scr) do
6091         for _, rg in pairs(maps) do
6092           if item.char >= rg[1] and item.char <= rg[2] then
6093             Babel.chr_to_loc[item.char] = lc
6094             toloc = lc
6095             break
6096           end
6097         end
6098       end
6099       % Treat composite chars in a different fashion, because they
6100       % 'inherit' the previous locale.
6101       if (item.char >= 0x0300 and item.char <= 0x036F) or
6102          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
6103          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
6104         Babel.chr_to_loc[item.char] = -2000
6105         toloc = -2000
6106       end
6107       if not toloc then
6108         Babel.chr_to_loc[item.char] = -1000
6109       end
6110     end
6111     if toloc == -2000 then
6112       toloc = toloc_save
6113     elseif toloc == -1000 then
6114       toloc = nil
6115     end
6116     if toloc and Babel.locale_props[toloc] and
6117        Babel.locale_props[toloc].letters and
6118        tex.getcatcode(item.char) \string~= 11 then
6119       toloc = nil
6120     end
6121     if toloc and Babel.locale_props[toloc].script
6122        and Babel.locale_props[node.get_attribute(item, LOCALE)].script
6123        and Babel.locale_props[toloc].script ==
6124        Babel.locale_props[node.get_attribute(item, LOCALE)].script then
6125       toloc = nil
6126     end
6127     if toloc then
6128       if Babel.locale_props[toloc].lg then
6129         item.lang = Babel.locale_props[toloc].lg
6130         node.set_attribute(item, LOCALE, toloc)
6131       end
6132       if Babel.locale_props[toloc]['/'..item.font] then
6133         item.font = Babel.locale_props[toloc]['/'..item.font]
6134       end
6135     end
6136     toloc_save = toloc
6137   elseif not inmath and item.id == 7 then % Apply recursively
6138     item.replace = item.replace and Babel.locale_map(item.replace)
6139     item.pre      = item.pre and Babel.locale_map(item.pre)
6140     item.post     = item.post and Babel.locale_map(item.post)
6141   elseif item.id == node.id'math' then
6142     inmath = (item.subtype == 0)
6143   end

```

```

6144 end
6145 return head
6146 end
6147 }

```

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

```

6148 \newcommand\babelcharproperty[1]{%
6149   \count@=#1\relax
6150   \ifvmode
6151     \expandafter\babel@chprop
6152   \else
6153     \babel@error{charproperty-only-vertical}{}{}{}%
6154   \fi}
6155 \newcommand\babel@chprop[3][\the\count@]{%
6156   \@tempcnta=#1\relax
6157   \babel@ifunset{babel@chprop@#2}% {unknown-char-property}
6158   {\babel@error{unknown-char-property}{}{#2}{}%
6159   {}%
6160   \loop
6161     \babel@cs{chprop@#2}{#3}%
6162     \ifnum\count@<\@tempcnta
6163       \advance\count@\@ne
6164     \repeat}
6165 \def\babel@chprop@direction#1{%
6166   \directlua{
6167     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6168     Babel.characters[\the\count@]['d'] = '#1'
6169   }}
6170 \let\babel@chprop@bc\babel@chprop@direction
6171 \def\babel@chprop@mirror#1{%
6172   \directlua{
6173     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6174     Babel.characters[\the\count@]['m'] = '\number#1'
6175   }}
6176 \let\babel@chprop@bmg\babel@chprop@mirror
6177 \def\babel@chprop@linebreak#1{%
6178   \directlua{
6179     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6180     Babel.cjk_characters[\the\count@]['c'] = '#1'
6181   }}
6182 \let\babel@chprop@lb\babel@chprop@linebreak
6183 \def\babel@chprop@locale#1{%
6184   \directlua{
6185     Babel.chr_to_loc = Babel.chr_to_loc or {}
6186     Babel.chr_to_loc[\the\count@] =
6187       \babel@ifblank{#1}{-1000}{\the\babel@cs{id@#1}}\space
6188   }}

```

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.

```

6189 \directlua{
6190   Babel.nohyphenation = \the\l@nohyphenation
6191 }

```

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

```

6192 \begingroup
6193 \catcode`\~ = 12
6194 \catcode`\% = 12
6195 \catcode`\& = 14
6196 \catcode`\| = 12
6197 \gdef\babelprehyphenation{%
6198   \@ifnextchar[{\babel@settransform{0}}{\babel@settransform{0}}{}
6199 \gdef\babelposthyphenation{%
6200   \@ifnextchar[{\babel@settransform{1}}{\babel@settransform{1}}{}
6201 \gdef\bbl@settransform#1[#2]#3#4#5{%
6202   \ifcase#1
6203     \bbl@activateprehyphen
6204   \or
6205     \bbl@activateposthyphen
6206   \fi
6207 \begingroup
6208   \def\babeltempa{\bbl@add@list\babeltempb}%
6209   \let\babeltempb\empty
6210   \def\bbl@tempa{#5}%
6211   \bbl@replace\bbl@tempa{,}{ ,}% TODO. Ugly trick to preserve {}
6212   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%
6213     \bbl@ifsamestring{##1}{remove}%
6214     {\bbl@add@list\babeltempb{nil}}}%
6215     {\directlua{
6216       local rep = {[##1]=}
6217       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6218       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6219       rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
6220       if #1 == 0 or #1 == 2 then
6221         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6222           'space = { ' .. '%2, %3, %4' .. ' }')
6223         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6224           'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6225         rep = rep:gsub('(kashida)%s*=%s*([%s,]*)', Babel.capture_kashida)
6226       else
6227         rep = rep:gsub(' (no)%s*=%s*([%s,]*)', Babel.capture_func)
6228         rep = rep:gsub(' (pre)%s*=%s*([%s,]*)', Babel.capture_func)
6229         rep = rep:gsub(' (post)%s*=%s*([%s,]*)', Babel.capture_func)
6230       end
6231       tex.print([[string\babeltempa{[]] .. rep .. [{}]])
6232     }}}%
6233 \bbl@foreach\babeltempb{%
6234   \bbl@forkv{##1}{%
6235     \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,&
6236     no,post,penalty,kashida,space,spacefactor,}%
6237     \ifin\else
6238     \bbl@error{bad-transform-option}{###1}{,}%
6239   \fi}}%
6240 \let\bbl@kv@attribute\relax
6241 \let\bbl@kv@label\relax
6242 \let\bbl@kv@fonts\empty
6243 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}%
6244 \ifx\bbl@kv@fonts\empty\else\bbl@settransform\fi
6245 \ifx\bbl@kv@attribute\relax
6246   \ifx\bbl@kv@label\relax\else
6247     \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}%
6248     \bbl@replace\bbl@kv@fonts{ ,}{ ,}%
6249     \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}%
6250   \count@\z@
6251   \def\bbl@elt##1##2##3{%
6252     \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}%
6253     {\bbl@ifsamestring{\bbl@kv@fonts}{##3}%
6254       {\count@\@ne}%

```



```

6255         {\bbl@error{font-conflict-transforms}{}}{}{}&%
6256     {}&%
6257     \bbl@transfont@list
6258     \ifnum\count@=\z@
6259         \bbl@exp{\global\\bbl@add\\bbl@transfont@list
6260             {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6261     \fi
6262     \bbl@ifunset{\bbl@kv@attribute}&%
6263     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6264     {}&%
6265     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6266 \fi
6267 \else
6268     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6269 \fi
6270 \directlua{
6271     local lbkr = Babel.linebreaking.replacements[#1]
6272     local u = unicode.utf8
6273     local id, attr, label
6274     if #1 == 0 then
6275         id = \the\csname bbl@id@#3\endcsname\space
6276     else
6277         id = \the\csname l@#3\endcsname\space
6278     end
6279     \ifx\bbl@kv@attribute\relax
6280         attr = -1
6281     \else
6282         attr = luatexbase.registernumber'\bbl@kv@attribute'
6283     \fi
6284     \ifx\bbl@kv@label\relax\else &% Same refs:
6285         label = [==[\bbl@kv@label]==]
6286     \fi
6287     &% Convert pattern:
6288     local patt = string.gsub([==[#4]==], '%s', '')
6289     if #1 == 0 then
6290         patt = string.gsub(patt, '|', ' ')
6291     end
6292     if not u.find(patt, '()', nil, true) then
6293         patt = '()' .. patt .. '()'
6294     end
6295     if #1 == 1 then
6296         patt = string.gsub(patt, '%(%)^', '^()')
6297         patt = string.gsub(patt, '%$$(%)', '()$')
6298     end
6299     patt = u.gsub(patt, '{(.)}',
6300         function (n)
6301             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6302         end)
6303     patt = u.gsub(patt, '{(%x%x%x%x+)}',
6304         function (n)
6305             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6306         end)
6307     lbkr[id] = lbkr[id] or {}
6308     table.insert(lbkr[id],
6309         { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6310 }&%
6311 \endgroup}
6312 \endgroup
6313 \let\bbl@transfont@list\@empty
6314 \def\bbl@settransfont{%
6315     \global\let\bbl@settransfont\relax % Execute only once
6316     \gdef\bbl@transfont{%
6317         \def\bbl@elt####1####2####3{%

```

```

6318 \bbl@ifblank{###3}%
6319 {\count@tw@}% Do nothing if no fonts
6320 {\count@z@
6321 \bbl@vforeach{###3}{%
6322 \def\bbl@tempd{#####1}%
6323 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6324 \ifx\bbl@tempd\bbl@tempe
6325 \count@one
6326 \else\ifx\bbl@tempd\bbl@transfam
6327 \count@one
6328 \fi\fi}%
6329 \ifcase\count@
6330 \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
6331 \or
6332 \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
6333 \fi}%
6334 \bbl@transfont@list}%
6335 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6336 \gdef\bbl@transfam{-unknown-}%
6337 \bbl@foreach\bbl@font@fams{%
6338 \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6339 \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6340 {\xdef\bbl@transfam{##1}}%
6341 {}}}
6342 \DeclareRobustCommand\enablelocaletransform[1]{%
6343 \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6344 {\bbl@error{transform-not-available}{#1}}}%
6345 {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6346 \DeclareRobustCommand\disablelocaletransform[1]{%
6347 \bbl@ifunset{\bbl@ATR@#1@\languagename @}%
6348 {\bbl@error{transform-not-available-b}{#1}}}%
6349 {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6350 \def\bbl@activateposthyphen{%
6351 \let\bbl@activateposthyphen\relax
6352 \directlua{
6353 require('babel-transforms.lua')
6354 Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6355 }}
6356 \def\bbl@activateprehyphen{%
6357 \let\bbl@activateprehyphen\relax
6358 \directlua{
6359 require('babel-transforms.lua')
6360 Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6361 }}

```

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.

```

6362 \newcommand\localeprehyphenation[1]{%
6363 \directlua{ Babel.string_prehyphenation([=#1]=], \the\localeid) }}

```

10.9 Bidi

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

```

6364 \def\bbl@activate@preotf{%
6365 \let\bbl@activate@preotf\relax % only once
6366 \directlua{
6367 Babel = Babel or {}
6368 %

```

```

6369 function Babel.pre_otfload_v(head)
6370   if Babel.numbers and Babel.digits_mapped then
6371     head = Babel.numbers(head)
6372   end
6373   if Babel.bidi_enabled then
6374     head = Babel.bidi(head, false, dir)
6375   end
6376   return head
6377 end
6378 %
6379 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6380   if Babel.numbers and Babel.digits_mapped then
6381     head = Babel.numbers(head)
6382   end
6383   if Babel.bidi_enabled then
6384     head = Babel.bidi(head, false, dir)
6385   end
6386   return head
6387 end
6388 %
6389 luatexbase.add_to_callback('pre_linebreak_filter',
6390   Babel.pre_otfload_v,
6391   'Babel.pre_otfload_v',
6392   luatexbase.priority_in_callback('pre_linebreak_filter',
6393     'luaotfload.node_processor') or nil)
6394 %
6395 luatexbase.add_to_callback('hpack_filter',
6396   Babel.pre_otfload_h,
6397   'Babel.pre_otfload_h',
6398   luatexbase.priority_in_callback('hpack_filter',
6399     'luaotfload.node_processor') or nil)
6400 }}

```

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=`.

```

6401 \breakafterdirmode=1
6402 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6403   \let\bbl@beforeforeign\leavevmode
6404   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6405   \RequirePackage{luatexbase}
6406   \bbl@activate@preotf
6407   \directlua{
6408     require('babel-data-bidi.lua')
6409     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6410       require('babel-bidi-basic.lua')
6411     \or
6412       require('babel-bidi-basic-r.lua')
6413     \fi}
6414   \newattribute\bbl@attr@dir
6415   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6416   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6417 \fi
6418 \chardef\bbl@thetextdir\z@
6419 \chardef\bbl@thepardir\z@
6420 \def\bbl@getluadir#1{%
6421   \directlua{
6422     if tex.#ldir == 'TLT' then
6423       tex.sprint('0')
6424     elseif tex.#ldir == 'TRT' then
6425       tex.sprint('1')
6426     end}}
6427 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl

```

```

6428 \ifcase#3\relax
6429   \ifcase\bbl@getluadir{#1}\relax\else
6430     #2 TLT\relax
6431   \fi
6432 \else
6433   \ifcase\bbl@getluadir{#1}\relax
6434     #2 TRT\relax
6435   \fi
6436 \fi}
6437% ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6438\def\bbl@thedir{0}
6439\def\bbl@textdir#1{%
6440  \bbl@setluadir{text}\textdir{#1}%
6441  \chardef\bbl@thetextdir#1\relax
6442  \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6443  \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6444\def\bbl@pardir#1{% Used twice
6445  \bbl@setluadir{par}\pardir{#1}%
6446  \chardef\bbl@thepardir#1\relax}
6447\def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6448\def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6449\def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6450 \ifnum\bbl@bidimode>\z@ % Any bidi=
6451   \def\bbl@insidemath{0}%
6452   \def\bbl@everymath{\def\bbl@insidemath{1}}
6453   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6454   \frozen@everymath\expandafter{%
6455     \expandafter\bbl@everymath\the\frozen@everymath}
6456   \frozen@everydisplay\expandafter{%
6457     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6458   \AtBeginDocument{
6459     \directlua{
6460       function Babel.math_box_dir(head)
6461         if not (token.get_macro('bbl@insidemath') == '0') then
6462           if Babel.hlist_has_bidi(head) then
6463             local d = node.new(node.id'dir')
6464             d.dir = '+TRT'
6465             node.insert_before(head, node.has_glyph(head), d)
6466             for item in node.traverse(head) do
6467               node.set_attribute(item,
6468                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6469             end
6470           end
6471         end
6472         return head
6473       end
6474       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6475         "Babel.math_box_dir", 0)
6476     }%
6477 \fi

```

10.10 Layout

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

Still, there are three areas deserving special attention, namely, tabular, math, and graphics, text and intrinsically left-to-right elements are intermingled. I’ve made some progress in graphics, but they’re essentially hacks; I’ve also made some progress in ‘tabular’, but when I decided to tackle math (both

standard math and ‘amsmath’) the nightmare began. I’m still not sure how ‘amsmath’ should be modified, but the main problem is that, boxes are “generic” containers that can hold text, math, and graphics (even at the same time; remember that inline math is included in the list of text nodes marked with ‘math’ (11) nodes too).

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

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

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

```

6478 \bbl@trace{Redefinitions for bidi layout}
6479 %
6480 <<(*More package options)>> ≡
6481 \chardef\bbl@eqnpos\z@
6482 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6483 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6484 <</More package options>>
6485 %
6486 \ifnum\bbl@bidimode>\z@ % Any bidi=
6487   \matheqdirmode\@ne % A luatex primitive
6488   \let\bbl@eqnodir\relax
6489   \def\bbl@eqdel{()}
6490   \def\bbl@eqnum{%
6491     {\normalfont\normalcolor
6492       \expandafter\@firstoftwo\bbl@eqdel
6493       \theequation
6494       \expandafter\@secondoftwo\bbl@eqdel}}
6495   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6496   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6497   \def\bbl@eqno@flip#1{%
6498     \ifdim\predisplaysize=-\maxdimen
6499       \eqno
6500       \hb@xt@.01pt{%
6501         \hb@xt@\displaywidth{\hss#1\glet\bbl@upset\@currentlabel}}\hss}%
6502     \else
6503       \leqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6504     \fi
6505     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6506   \def\bbl@leqno@flip#1{%
6507     \ifdim\predisplaysize=-\maxdimen
6508       \leqno
6509       \hb@xt@.01pt{%
6510         \hss\hb@xt@\displaywidth{\{#1\glet\bbl@upset\@currentlabel}\hss}}%
6511     \else
6512       \eqno\hbox{#1\glet\bbl@upset\@currentlabel}%
6513     \fi
6514     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6515   \AtBeginDocument{%
6516     \ifx\bbl@noamsmath\relax\else
6517     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6518       \AddToHook{env/equation/begin}{%
6519         \ifnum\bbl@thetextdir>\z@
6520           \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6521           \let\@eqnnum\bbl@eqnum
6522           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6523           \chardef\bbl@thetextdir\z@
6524           \bbl@add\normalfont{\bbl@eqnodir}%
6525           \ifcase\bbl@eqnpos
6526             \let\bbl@puteqno\bbl@eqno@flip
6527           \or
6528             \let\bbl@puteqno\bbl@leqno@flip

```

```

6529     \fi
6530     \fi}%
6531 \ifnum\bbled@eqnpos=\tw@ \else
6532 \def\endequation{\bbled@puteqno{\@eqnnum}$\@ignoretrue}%
6533 \fi
6534 \AddToHook{env/eqnarray/begin}{%
6535 \ifnum\bbled@thetextdir>\z@
6536 \def\bbled@mathboxdir{\def\bbled@insidemath{1}}%
6537 \edef\bbled@eqnodir{\noexpand\bbled@textdir{\the\bbled@thetextdir}}%
6538 \chardef\bbled@thetextdir\z@
6539 \bbled@add\normalfont{\bbled@eqnodir}%
6540 \ifnum\bbled@eqnpos=\@ne
6541 \def\@eqnnum{%
6542 \setbox\z@\hbox{\bbled@eqnum}%
6543 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6544 \else
6545 \let\@eqnnum\bbled@eqnum
6546 \fi
6547 \fi}
6548 % Hack. YA luatex bug?:
6549 \expandafter\bbled@sreplace\csname\endcsname{\eqno\kern.001pt$}$%
6550 \else % amstex
6551 \bbled@exp{% Hack to hide maybe undefined conditionals:
6552 \chardef\bbled@eqnpos=0%
6553 \<iftagsleft>1<else>\<if@fleqn>2<fi>\<fi>\relax}%
6554 \ifnum\bbled@eqnpos=\@ne
6555 \let\bbled@ams@lap\hbox
6556 \else
6557 \let\bbled@ams@lap\llap
6558 \fi
6559 \ExplSyntaxOn % Required by \bbled@sreplace with \intertext@
6560 \bbled@sreplace\intertext@{\normalbaselines}%
6561 {\normalbaselines
6562 \ifx\bbled@eqnodir\relax\else\bbled@pardir\@ne\bbled@eqnodir\fi}%
6563 \ExplSyntaxOff
6564 \def\bbled@ams@tagbox#1#2{#1{\bbled@eqnodir#2}}% #1=hbox|@lap|flip
6565 \ifx\bbled@ams@lap\hbox % leqno
6566 \def\bbled@ams@flip#1{%
6567 \hbox to 0.01pt{\hss\hbox to\displaywidth{\#1\hss}}}%
6568 \else % eqno
6569 \def\bbled@ams@flip#1{%
6570 \hbox to 0.01pt{\hbox to\displaywidth{\hss\#1}\hss}}%
6571 \fi
6572 \def\bbled@ams@preset#1{%
6573 \def\bbled@mathboxdir{\def\bbled@insidemath{1}}%
6574 \ifnum\bbled@thetextdir>\z@
6575 \edef\bbled@eqnodir{\noexpand\bbled@textdir{\the\bbled@thetextdir}}%
6576 \bbled@sreplace\textdef@{\hbox}{\bbled@ams@tagbox\hbox}%
6577 \bbled@sreplace\maketag@@@{\hbox}{\bbled@ams@tagbox#1}%
6578 \fi}%
6579 \ifnum\bbled@eqnpos=\tw@ \else
6580 \def\bbled@ams@equation{%
6581 \def\bbled@mathboxdir{\def\bbled@insidemath{1}}%
6582 \ifnum\bbled@thetextdir>\z@
6583 \edef\bbled@eqnodir{\noexpand\bbled@textdir{\the\bbled@thetextdir}}%
6584 \chardef\bbled@thetextdir\z@
6585 \bbled@add\normalfont{\bbled@eqnodir}%
6586 \ifcase\bbled@eqnpos
6587 \def\veqno##1##2{\bbled@eqno@flip{##1##2}}%
6588 \or
6589 \def\veqno##1##2{\bbled@leqno@flip{##1##2}}%
6590 \fi
6591 \fi}%

```

```

6592 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6593 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6594 \fi
6595 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6596 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6597 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6598 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6599 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6600 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6601 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6602 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6603 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6604 % Hackish, for proper alignment. Don't ask me why it works!:
6605 \bbl@exp{% Avoid a 'visible' conditional
6606   \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}>\<fi>}%
6607   \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{}>\<fi>}}%
6608 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6609 \AddToHook{env/split/before}{%
6610   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6611   \ifnum\bbl@thetextdir>\z@
6612     \bbl@ifsamestring\currentenv{equation}%
6613     {\ifx\bbl@ams@lap\hbox % leqno
6614       \def\bbl@ams@flip#1{%
6615         \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6616       \else
6617         \def\bbl@ams@flip#1{%
6618           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#{1}}}}%
6619         \fi}%
6620     }%
6621   \fi}%
6622 \fi\fi}
6623 \fi
6624 \def\bbl@provide@extra#1{%
6625   % == Counters: mapdigits ==
6626   % Native digits
6627   \ifx\bbl@KVP@mapdigits\@nnil\else
6628     \bbl@ifunset{bbl@dgnat@\language\name}{}%
6629     {\RequirePackage{luatexbase}%
6630      \bbl@activate@preotf
6631      \directlua{
6632        Babel = Babel or {} %%% -> presets in luababel
6633        Babel.digits_mapped = true
6634        Babel.digits = Babel.digits or {}
6635        Babel.digits[\the\localeid] =
6636          table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6637        if not Babel.numbers then
6638          function Babel.numbers(head)
6639            local LOCALE = Babel.attr_locale
6640            local GLYPH = node.id'glyph'
6641            local inmath = false
6642            for item in node.traverse(head) do
6643              if not inmath and item.id == GLYPH then
6644                local temp = node.get_attribute(item, LOCALE)
6645                if Babel.digits[temp] then
6646                  local chr = item.char
6647                  if chr > 47 and chr < 58 then
6648                    item.char = Babel.digits[temp][chr-47]
6649                  end
6650                end
6651              elseif item.id == node.id'math' then
6652                inmath = (item.subtype == 0)
6653              end
6654            end

```

```

6655         return head
6656     end
6657 end
6658 }}%
6659 \fi
6660 % == transforms ==
6661 \ifx\bbI@KVP@transforms\@nnil\else
6662   \def\bbI@elt##1##2##3{%
6663     \in@{${transforms.}}{##1}%
6664     \ifin@
6665       \def\bbI@tempa{##1}%
6666       \bbI@replace\bbI@tempa{transforms.}{}%
6667       \bbI@carg\bbI@transforms{babel\bbI@tempa}{##2}{##3}%
6668     \fi}%
6669   \csname bbl@inidata@\language\endcsname
6670   \bbI@release@transforms\relax % \relax closes the last item.
6671 \fi}
6672 % Start tabular here:
6673 \def\localerestoredirs{%
6674   \ifcase\bbI@thetextdir
6675     \ifnum\textdirection=\z@\else\textdir TLT\fi
6676   \else
6677     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6678   \fi
6679   \ifcase\bbI@thepardir
6680     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6681   \else
6682     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6683   \fi}
6684 \IfBabelLayout{tabular}%
6685   {\chardef\bbI@tabular@mode\tw@}% All RTL
6686   {\IfBabelLayout{notabular}%
6687     {\chardef\bbI@tabular@mode\z@}%
6688     {\chardef\bbI@tabular@mode\@ne}% Mixed, with LTR cols
6689   \ifnum\bbI@bidimode>\@ne % Any lua bidi= except default=1
6690   \ifcase\bbI@tabular@mode\or % 1
6691     \let\bbI@parabefore\relax
6692     \AddToHook{para/before}{\bbI@parabefore}
6693     \AtBeginDocument{%
6694       \bbI@replace\@tabular{$}{${}%
6695       \def\bbI@insidemath{0}%
6696       \def\bbI@parabefore{\localerestoredirs}}%
6697     \ifnum\bbI@tabular@mode=\@ne
6698       \bbI@ifunset{\@tabclassz}{}%
6699       \bbI@exp{% Hide conditionals
6700         \\\bbI@sreplace\\\@tabclassz
6701         {\<ifcase>\\\@chnum}%
6702         {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6703       \@ifpackageloaded{colortbl}%
6704       {\bbI@sreplace\@classz
6705        {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6706       {\@ifpackageloaded{array}%
6707        {\bbI@exp{% Hide conditionals
6708          \\\bbI@sreplace\\\@classz
6709          {\<ifcase>\\\@chnum}%
6710          {\bgroup\\localerestoredirs\<ifcase>\\\@chnum}%
6711          \\\bbI@sreplace\\\@classz
6712          {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6713        {}}%
6714     \fi}%
6715   \or % 2
6716     \let\bbI@parabefore\relax
6717     \AddToHook{para/before}{\bbI@parabefore}%

```



```

6718 \AtBeginDocument{%
6719 \ifpackageloaded{colortbl}%
6720 {\bbl@replace\@tabular{$}{$}%
6721 \def\bbl@insidemath{0}%
6722 \def\bbl@parabefore{\localerestoredirs}}%
6723 \bbl@sreplace\@classz
6724 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6725 }%
6726 \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.

```

6727 \AtBeginDocument{%
6728 \ifpackageloaded{multicol}%
6729 {\toks@expandafter{\multi@column@out}%
6730 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6731 }%
6732 \ifpackageloaded{paracol}%
6733 {\edef\pcol@output{%
6734 \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6735 }%
6736 \fi
6737 \ifx\bbl@opt@layout@nnil\endinput\fi % if no layout

```

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

```

6738 \ifnum\bbl@bidimode>\z@ % Any bidi=
6739 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6740 \bbl@exp{%
6741 \def\\bbl@insidemath{0}%
6742 \mathdir\the\bodydir
6743 #1% Once entered in math, set boxes to restore values
6744 \<ifmmode>%
6745 \everyvbox{%
6746 \the\everyvbox
6747 \bodydir\the\bodydir
6748 \mathdir\the\mathdir
6749 \everyhbox{\the\everyhbox}%
6750 \everyvbox{\the\everyvbox}}%
6751 \everyhbox{%
6752 \the\everyhbox
6753 \bodydir\the\bodydir
6754 \mathdir\the\mathdir
6755 \everyhbox{\the\everyhbox}%
6756 \everyvbox{\the\everyvbox}}%
6757 \<fi>}}%
6758 \def\@hangfrom#1{%
6759 \setbox\@tempboxa\hbox{#1}%
6760 \hangindent\wd\@tempboxa
6761 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6762 \shapemode\@ne
6763 \fi
6764 \noindent\box\@tempboxa}
6765 \fi
6766 \IfBabelLayout{tabular}
6767 {\let\bbl@OL\@tabular\@tabular
6768 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6769 \let\bbl@NL\@tabular\@tabular
6770 \AtBeginDocument{%
6771 \ifx\bbl@NL\@tabular\@tabular\else
6772 \bbl@exp{\\in{\bbl@nextfake}{\@tabular}}}%

```

```

6773     \ifin@else
6774     \bbl@replace\@tabular{$}\bbl@nextfake$}%
6775     \fi
6776     \let\bbl@NL@\@tabular\@tabular
6777     \fi}}
6778     {}
6779 \IfBabelLayout{lists}
6780 {\let\bbl@OL@list\list
6781  \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6782  \let\bbl@NL@list\list
6783  \def\bbl@listparshape#1#2#3{%
6784   \parshape #1 #2 #3 %
6785   \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6786   \shapemode\tw@
6787   \fi}}
6788 {}
6789 \IfBabelLayout{graphics}
6790 {\let\bbl@pictresetdir\relax
6791  \def\bbl@pictsetdir#1{%
6792   \ifcase\bbl@thetextdir
6793   \let\bbl@pictresetdir\relax
6794   \else
6795   \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6796   \or\textdir TLT
6797   \else\bodydir TLT \textdir TLT
6798   \fi
6799   % \text\par)dir required in pgf:
6800   \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6801   \fi}%
6802  \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6803  \directlua{
6804   Babel.get_picture_dir = true
6805   Babel.picture_has_bidi = 0
6806   %
6807   function Babel.picture_dir (head)
6808     if not Babel.get_picture_dir then return head end
6809     if Babel.hlist_has_bidi(head) then
6810       Babel.picture_has_bidi = 1
6811     end
6812     return head
6813   end
6814   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6815     "Babel.picture_dir")
6816  }%
6817  \AtBeginDocument{%
6818   \def\LS@rot{%
6819    \setbox\@outputbox\vbox{%
6820     \hbox dir TLT{\rotatebox{90}\box\@outputbox}}}%
6821   \long\def\put(#1,#2)#3{%
6822    \@killglue
6823    % Try:
6824    \ifx\bbl@pictresetdir\relax
6825    \def\bbl@tempc{0}%
6826    \else
6827    \directlua{
6828     Babel.get_picture_dir = true
6829     Babel.picture_has_bidi = 0
6830    }%
6831    \setbox\z@\hb@xt@\z@{%
6832     \@defaultunitsset\@tempdimc{#1}\unitlength
6833     \kern\@tempdimc
6834     #3\hss}% TODO: #3 executed twice (below). That's bad.
6835    \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%

```

```

6836 \fi
6837 % Do:
6838 \@defaultunitsset\@tempdimc{#2}\unitlength
6839 \raise\@tempdimc\hbext\z@{\%
6840 \@defaultunitsset\@tempdimc{#1}\unitlength
6841 \kern\@tempdimc
6842 {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6843 \ignorespaces}%
6844 \MakeRobust\put}%
6845 \AtBeginDocument
6846 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir@gobble}%
6847 \ifx\pgfpicture\undefined\else % TODO. Allow deactivate?
6848 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir@ne}%
6849 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6850 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6851 \fi
6852 \ifx\tikzpicture\undefined\else
6853 \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
6854 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6855 \bbbl@sreplace\tikz{\begingroup}{\begingroup\bbbl@pictsetdir\tw@}%
6856 \fi
6857 \ifx\tcolorbox\undefined\else
6858 \def\tcb@drawing@env@begin{%
6859 \csname tcb@before@\tcb@split@state\endcsname
6860 \bbbl@pictsetdir\tw@
6861 \begin{\kv tcb@graphenv}%
6862 \tcb@bbdraw%
6863 \tcb@apply@graph@patches
6864 }%
6865 \def\tcb@drawing@env@end{%
6866 \end{\kv tcb@graphenv}%
6867 \bbbl@pictresetdir
6868 \csname tcb@after@\tcb@split@state\endcsname
6869 }%
6870 \fi
6871 }}
6872 {}

```

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.

```

6873 \IfBabelLayout{counters*}%
6874 {\bbbl@add\bbbl@opt@layout{.counters.}%
6875 \directlua{
6876 \luaexec{
6877 \luaexec{
6878 }}}}
6879 \IfBabelLayout{counters}%
6880 {\let\bbbl@OL@@textsuperscript\@textsuperscript
6881 \bbbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6882 \let\bbbl@latinarabic=\@arabic
6883 \let\bbbl@OL@@arabic\@arabic
6884 \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
6885 \@ifpackagewith{babel}{bidi=default}%
6886 {\let\bbbl@asciroman=\@roman
6887 \let\bbbl@OL@@roman\@roman
6888 \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
6889 \let\bbbl@asciiRoman=\@Roman
6890 \let\bbbl@OL@@roman\@Roman
6891 \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
6892 \let\bbbl@OL@labelenumii\labelenumii
6893 \def\labelenumii{\theenumii}%
6894 \let\bbbl@OL@p@enumiii\p@enumiii

```

```

6895 \def\p@enumiii{\p@enumii}\theenumii({}){}{}{}
6896 <<Footnote changes>>
6897 \IfBabelLayout{footnotes}%
6898 {\let\bbl@OL@footnote\footnote
6899 \BabelFootnote\footnote\languagename{}{}%
6900 \BabelFootnote\localfootnote\languagename{}{}%
6901 \BabelFootnote\mainfootnote{}{}{}}
6902 {}

```

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

```

6903 \IfBabelLayout{extras}%
6904 {\bbl@ncarg\let\bbl@OL@underline{underline }%
6905 \bbl@carg\bbl@sreplace{underline }%
6906 {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
6907 \bbl@carg\bbl@sreplace{underline }%
6908 {\m@th$}{\m@th$\egroup}%
6909 \let\bbl@OL@LaTeXe\LaTeXe
6910 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6911 \if b\expandafter\@car\f@series\@nil\boldmath\fi
6912 \babelsublr}%
6913 \LaTeXe\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}
6914 {}
6915 </luatex>

```

10.11 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.

```

6916 (*transforms)
6917 Babel.linebreaking.replacements = {}
6918 Babel.linebreaking.replacements[0] = {} -- pre
6919 Babel.linebreaking.replacements[1] = {} -- post
6920
6921 -- Discretionaries contain strings as nodes
6922 function Babel.str_to_nodes(fn, matches, base)
6923   local n, head, last
6924   if fn == nil then return nil end
6925   for s in string.utfvalues(fn(matches)) do
6926     if base.id == 7 then
6927       base = base.replace
6928     end
6929     n = node.copy(base)
6930     n.char = s
6931     if not head then
6932       head = n
6933     else
6934       last.next = n
6935     end
6936     last = n
6937   end
6938   return head
6939 end
6940

```

```

6941 Babel.fetch_subtext = {}
6942
6943 Babel.ignore_pre_char = function(node)
6944   return (node.lang == Babel.nohyphenation)
6945 end
6946
6947 -- Merging both functions doesn't seem feasible, because there are too
6948 -- many differences.
6949 Babel.fetch_subtext[0] = function(head)
6950   local word_string = ''
6951   local word_nodes = {}
6952   local lang
6953   local item = head
6954   local inmath = false
6955
6956   while item do
6957     if item.id == 11 then
6958       inmath = (item.subtype == 0)
6959     end
6960
6961     if inmath then
6962       -- pass
6963     end
6964
6965     elseif item.id == 29 then
6966       local locale = node.get_attribute(item, Babel.attr_locale)
6967
6968       if lang == locale or lang == nil then
6969         lang = locale
6970         if Babel.ignore_pre_char(item) then
6971           word_string = word_string .. Babel.us_char
6972         else
6973           word_string = word_string .. unicode.utf8.char(item.char)
6974         end
6975         word_nodes[#word_nodes+1] = item
6976       else
6977         break
6978       end
6979
6980       elseif item.id == 12 and item.subtype == 13 then
6981         word_string = word_string .. ' '
6982         word_nodes[#word_nodes+1] = item
6983
6984         -- Ignore leading unrecognized nodes, too.
6985         elseif word_string ~= '' then
6986           word_string = word_string .. Babel.us_char
6987           word_nodes[#word_nodes+1] = item -- Will be ignored
6988         end
6989
6990         item = item.next
6991       end
6992
6993       -- Here and above we remove some trailing chars but not the
6994       -- corresponding nodes. But they aren't accessed.
6995       if word_string:sub(-1) == ' ' then
6996         word_string = word_string:sub(1,-2)
6997       end
6998       word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6999       return word_string, word_nodes, item, lang
7000 end
7001
7002 Babel.fetch_subtext[1] = function(head)
7003   local word_string = ''

```

```

7004 local word_nodes = {}
7005 local lang
7006 local item = head
7007 local inmath = false
7008
7009 while item do
7010
7011     if item.id == 11 then
7012         inmath = (item.subtype == 0)
7013     end
7014
7015     if inmath then
7016         -- pass
7017
7018     elseif item.id == 29 then
7019         if item.lang == lang or lang == nil then
7020             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
7021                 lang = lang or item.lang
7022                 word_string = word_string .. unicode.utf8.char(item.char)
7023                 word_nodes[#word_nodes+1] = item
7024             end
7025         else
7026             break
7027         end
7028
7029     elseif item.id == 7 and item.subtype == 2 then
7030         word_string = word_string .. '='
7031         word_nodes[#word_nodes+1] = item
7032
7033     elseif item.id == 7 and item.subtype == 3 then
7034         word_string = word_string .. '|'
7035         word_nodes[#word_nodes+1] = item
7036
7037         -- (1) Go to next word if nothing was found, and (2) implicitly
7038         -- remove leading USs.
7039     elseif word_string == '' then
7040         -- pass
7041
7042         -- This is the responsible for splitting by words.
7043     elseif (item.id == 12 and item.subtype == 13) then
7044         break
7045
7046     else
7047         word_string = word_string .. Babel.us_char
7048         word_nodes[#word_nodes+1] = item -- Will be ignored
7049     end
7050
7051     item = item.next
7052 end
7053
7054 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
7055 return word_string, word_nodes, item, lang
7056 end
7057
7058 function Babel.pre_hyphenate_replace(head)
7059     Babel.hyphenate_replace(head, 0)
7060 end
7061
7062 function Babel.post_hyphenate_replace(head)
7063     Babel.hyphenate_replace(head, 1)
7064 end
7065
7066 Babel.us_char = string.char(31)

```

```

7067
7068 function Babel.hyphenate_replace(head, mode)
7069   local u = unicode.utf8
7070   local lbkr = Babel.linebreaking.replacements[mode]
7071
7072   local word_head = head
7073
7074   while true do -- for each subtext block
7075
7076     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
7077
7078     if Babel.debug then
7079       print()
7080       print((mode == 0) and '@@@<' or '@@@>', w)
7081     end
7082
7083     if nw == nil and w == '' then break end
7084
7085     if not lang then goto next end
7086     if not lbkr[lang] then goto next end
7087
7088     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
7089     -- loops are nested.
7090     for k=1, #lbkr[lang] do
7091       local p = lbkr[lang][k].pattern
7092       local r = lbkr[lang][k].replace
7093       local attr = lbkr[lang][k].attr or -1
7094
7095       if Babel.debug then
7096         print('*****', p, mode)
7097       end
7098
7099       -- This variable is set in some cases below to the first *byte*
7100       -- after the match, either as found by u.match (faster) or the
7101       -- computed position based on sc if w has changed.
7102       local last_match = 0
7103       local step = 0
7104
7105       -- For every match.
7106       while true do
7107         if Babel.debug then
7108           print('====')
7109         end
7110         local new -- used when inserting and removing nodes
7111
7112         local matches = { u.match(w, p, last_match) }
7113
7114         if #matches < 2 then break end
7115
7116         -- Get and remove empty captures (with ()'s, which return a
7117         -- number with the position), and keep actual captures
7118         -- (from (...)), if any, in matches.
7119         local first = table.remove(matches, 1)
7120         local last = table.remove(matches, #matches)
7121         -- Non re-fetched substrings may contain \31, which separates
7122         -- subsubstrings.
7123         if string.find(w:sub(first, last-1), Babel.us_char) then break end
7124
7125         local save_last = last -- with A()BC()D, points to D
7126
7127         -- Fix offsets, from bytes to unicode. Explained above.
7128         first = u.len(w:sub(1, first-1)) + 1
7129         last = u.len(w:sub(1, last-1)) -- now last points to C

```

```

7130
7131 -- This loop stores in a small table the nodes
7132 -- corresponding to the pattern. Used by 'data' to provide a
7133 -- predictable behavior with 'insert' (w_nodes is modified on
7134 -- the fly), and also access to 'remove'd nodes.
7135 local sc = first-1 -- Used below, too
7136 local data_nodes = {}
7137
7138 local enabled = true
7139 for q = 1, last-first+1 do
7140     data_nodes[q] = w_nodes[sc+q]
7141     if enabled
7142         and attr > -1
7143         and not node.has_attribute(data_nodes[q], attr)
7144     then
7145         enabled = false
7146     end
7147 end
7148
7149 -- This loop traverses the matched substring and takes the
7150 -- corresponding action stored in the replacement list.
7151 -- sc = the position in substr nodes / string
7152 -- rc = the replacement table index
7153 local rc = 0
7154
7155 while rc < last-first+1 do -- for each replacement
7156     if Babel.debug then
7157         print('.....', rc + 1)
7158     end
7159     sc = sc + 1
7160     rc = rc + 1
7161
7162     if Babel.debug then
7163         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7164         local ss = ''
7165         for itt in node.traverse(head) do
7166             if itt.id == 29 then
7167                 ss = ss .. unicode.utf8.char(itt.char)
7168             else
7169                 ss = ss .. '{' .. itt.id .. '}'
7170             end
7171         end
7172         print('*****', ss)
7173     end
7174
7175     local crep = r[rc]
7176     local item = w_nodes[sc]
7177     local item_base = item
7178     local placeholder = Babel.us_char
7179     local d
7180
7181     if crep and crep.data then
7182         item_base = data_nodes[crep.data]
7183     end
7184
7185     if crep then
7186         step = crep.step or 0
7187     end
7188
7189     if (not enabled) or (crep and next(crep) == nil) then -- = {}
7190         last_match = save_last -- Optimization
7191         goto next
7192

```



```

7193
7194 elseif crep == nil or crep.remove then
7195     node.remove(head, item)
7196     table.remove(w_nodes, sc)
7197     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7198     sc = sc - 1 -- Nothing has been inserted.
7199     last_match = utf8.offset(w, sc+1+step)
7200     goto next
7201
7202 elseif crep and crep.kashida then -- Experimental
7203     node.set_attribute(item,
7204         Babel.attr_kashida,
7205         crep.kashida)
7206     last_match = utf8.offset(w, sc+1+step)
7207     goto next
7208
7209 elseif crep and crep.string then
7210     local str = crep.string(matches)
7211     if str == '' then -- Gather with nil
7212         node.remove(head, item)
7213         table.remove(w_nodes, sc)
7214         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7215         sc = sc - 1 -- Nothing has been inserted.
7216     else
7217         local loop_first = true
7218         for s in string.utfvalues(str) do
7219             d = node.copy(item_base)
7220             d.char = s
7221             if loop_first then
7222                 loop_first = false
7223                 head, new = node.insert_before(head, item, d)
7224                 if sc == 1 then
7225                     word_head = head
7226                 end
7227                 w_nodes[sc] = d
7228                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7229             else
7230                 sc = sc + 1
7231                 head, new = node.insert_before(head, item, d)
7232                 table.insert(w_nodes, sc, new)
7233                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7234             end
7235             if Babel.debug then
7236                 print('.....', 'str')
7237                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7238             end
7239             end -- for
7240             node.remove(head, item)
7241         end -- if ''
7242         last_match = utf8.offset(w, sc+1+step)
7243         goto next
7244
7245 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7246     d = node.new(7, 3) -- (disc, regular)
7247     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7248     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7249     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7250     d.attr = item_base.attr
7251     if crep.pre == nil then -- TeXbook p96
7252         d.penalty = crep.penalty or tex.hyphenpenalty
7253     else
7254         d.penalty = crep.penalty or tex.exhyphenpenalty
7255     end

```

```

7256         placeholder = '|'
7257         head, new = node.insert_before(head, item, d)
7258
7259     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7260         -- ERROR
7261
7262     elseif crep and crep.penalty then
7263         d = node.new(14, 0) -- (penalty, userpenalty)
7264         d.attr = item_base.attr
7265         d.penalty = crep.penalty
7266         head, new = node.insert_before(head, item, d)
7267
7268     elseif crep and crep.space then
7269         -- 655360 = 10 pt = 10 * 65536 sp
7270         d = node.new(12, 13) -- (glue, spaceskip)
7271         local quad = font.getfont(item_base.font).size or 655360
7272         node.setglue(d, crep.space[1] * quad,
7273                     crep.space[2] * quad,
7274                     crep.space[3] * quad)
7275         if mode == 0 then
7276             placeholder = ' '
7277         end
7278         head, new = node.insert_before(head, item, d)
7279
7280     elseif crep and crep.spacefactor then
7281         d = node.new(12, 13) -- (glue, spaceskip)
7282         local base_font = font.getfont(item_base.font)
7283         node.setglue(d,
7284                     crep.spacefactor[1] * base_font.parameters['space'],
7285                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
7286                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
7287         if mode == 0 then
7288             placeholder = ' '
7289         end
7290         head, new = node.insert_before(head, item, d)
7291
7292     elseif mode == 0 and crep and crep.space then
7293         -- ERROR
7294
7295     end -- ie replacement cases
7296
7297     -- Shared by disc, space and penalty.
7298     if sc == 1 then
7299         word_head = head
7300     end
7301     if crep.insert then
7302         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7303         table.insert(w_nodes, sc, new)
7304         last = last + 1
7305     else
7306         w_nodes[sc] = d
7307         node.remove(head, item)
7308         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7309     end
7310
7311     last_match = utf8.offset(w, sc+1+step)
7312
7313     ::next::
7314
7315 end -- for each replacement
7316
7317 if Babel.debug then
7318     print('.....', '/')

```

```

7319         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7320     end
7321
7322     end -- for match
7323
7324     end -- for patterns
7325
7326     ::next::
7327     word_head = nw
7328 end -- for substring
7329 return head
7330 end
7331
7332 -- This table stores capture maps, numbered consecutively
7333 Babel.capture_maps = {}
7334
7335 -- The following functions belong to the next macro
7336 function Babel.capture_func(key, cap)
7337     local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
7338     local cnt
7339     local u = unicode.utf8
7340     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7341     if cnt == 0 then
7342         ret = u.gsub(ret, '{(%X%X%X%X+)}',
7343             function (n)
7344                 return u.char(tonumber(n, 16))
7345             end)
7346     end
7347     ret = ret:gsub("%[%[%]]%.%", '')
7348     ret = ret:gsub("%.%.%[%[%]]", '')
7349     return key .. [=function(m) return ]] .. ret .. [=end]]
7350 end
7351
7352 function Babel.capt_map(from, mapno)
7353     return Babel.capture_maps[mapno][from] or from
7354 end
7355
7356 -- Handle the {n|abc|ABC} syntax in captures
7357 function Babel.capture_func_map(capno, from, to)
7358     local u = unicode.utf8
7359     from = u.gsub(from, '{(%X%X%X%X+)}',
7360         function (n)
7361             return u.char(tonumber(n, 16))
7362         end)
7363     to = u.gsub(to, '{(%X%X%X%X+)}',
7364         function (n)
7365             return u.char(tonumber(n, 16))
7366         end)
7367     local froms = {}
7368     for s in string.utfcharacters(from) do
7369         table.insert(froms, s)
7370     end
7371     local cnt = 1
7372     table.insert(Babel.capture_maps, {})
7373     local mlen = table.getn(Babel.capture_maps)
7374     for s in string.utfcharacters(to) do
7375         Babel.capture_maps[mlen][froms[cnt]] = s
7376         cnt = cnt + 1
7377     end
7378     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7379         (mlen) .. ").." .. "["
7380 end
7381

```

```

7382 -- Create/Extend reversed sorted list of kashida weights:
7383 function Babel.capture_kashida(key, wt)
7384   wt = tonumber(wt)
7385   if Babel.kashida_wts then
7386     for p, q in ipairs(Babel.kashida_wts) do
7387       if wt == q then
7388         break
7389       elseif wt > q then
7390         table.insert(Babel.kashida_wts, p, wt)
7391         break
7392       elseif table.getn(Babel.kashida_wts) == p then
7393         table.insert(Babel.kashida_wts, wt)
7394       end
7395     end
7396   else
7397     Babel.kashida_wts = { wt }
7398   end
7399   return 'kashida = ' .. wt
7400 end
7401
7402 -- Experimental: applies prehyphenation transforms to a string (letters
7403 -- and spaces).
7404 function Babel.string_prehyphenation(str, locale)
7405   local n, head, last, res
7406   head = node.new(8, 0) -- dummy (hack just to start)
7407   last = head
7408   for s in string.utfvalues(str) do
7409     if s == 20 then
7410       n = node.new(12, 0)
7411     else
7412       n = node.new(29, 0)
7413       n.char = s
7414     end
7415     node.set_attribute(n, Babel.attr_locale, locale)
7416     last.next = n
7417     last = n
7418   end
7419   head = Babel.hyphenate_replace(head, 0)
7420   res = ''
7421   for n in node.traverse(head) do
7422     if n.id == 12 then
7423       res = res .. ' '
7424     elseif n.id == 29 then
7425       res = res .. unicode.utf8.char(n.char)
7426     end
7427   end
7428   tex.print(res)
7429 end
7430 </transforms>

```

10.12 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.

```
7431 (*basic-r)
7432 Babel = Babel or {}
7433
7434 Babel.bidi_enabled = true
7435
7436 require('babel-data-bidi.lua')
7437
7438 local characters = Babel.characters
7439 local ranges = Babel.ranges
7440
7441 local DIR = node.id("dir")
7442
7443 local function dir_mark(head, from, to, outer)
7444   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7445   local d = node.new(DIR)
7446   d.dir = '+' .. dir
7447   node.insert_before(head, from, d)
7448   d = node.new(DIR)
7449   d.dir = '-' .. dir
7450   node.insert_after(head, to, d)
7451 end
7452
7453 function Babel.bidi(head, ispar)
7454   local first_n, last_n          -- first and last char with nums
7455   local last_es                  -- an auxiliary 'last' used with nums
7456   local first_d, last_d          -- first and last char in L/R block
7457   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):

```
7458   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7459   local strong_lr = (strong == 'l') and 'l' or 'r'
7460   local outer = strong
7461
7462   local new_dir = false
7463   local first_dir = false
```

```

7464 local inmath = false
7465
7466 local last_lr
7467
7468 local type_n = ''
7469
7470 for item in node.traverse(head) do
7471
7472   -- three cases: glyph, dir, otherwise
7473   if item.id == node.id'glyph'
7474     or (item.id == 7 and item.subtype == 2) then
7475
7476     local itemchar
7477     if item.id == 7 and item.subtype == 2 then
7478       itemchar = item.replace.char
7479     else
7480       itemchar = item.char
7481     end
7482     local chardata = characters[itemchar]
7483     dir = chardata and chardata.d or nil
7484     if not dir then
7485       for nn, et in ipairs(ranges) do
7486         if itemchar < et[1] then
7487           break
7488         elseif itemchar <= et[2] then
7489           dir = et[3]
7490           break
7491         end
7492       end
7493     end
7494     dir = dir or 'l'
7495     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).

```

7496   if new_dir then
7497     attr_dir = 0
7498     for at in node.traverse(item.attr) do
7499       if at.number == Babel.attr_dir then
7500         attr_dir = at.value & 0x3
7501       end
7502     end
7503     if attr_dir == 1 then
7504       strong = 'r'
7505     elseif attr_dir == 2 then
7506       strong = 'al'
7507     else
7508       strong = 'l'
7509     end
7510     strong_lr = (strong == 'l') and 'l' or 'r'
7511     outer = strong_lr
7512     new_dir = false
7513   end
7514
7515   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7516   dir_real = dir -- We need dir_real to set strong below
7517   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:

```

7518     if strong == 'al' then
7519         if dir == 'en' then dir = 'an' end           -- W2
7520         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7521         strong_lr = 'r'                               -- W3
7522     end

```

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

```

7523     elseif item.id == node.id'dir' and not inmath then
7524         new_dir = true
7525         dir = nil
7526     elseif item.id == node.id'math' then
7527         inmath = (item.subtype == 0)
7528     else
7529         dir = nil          -- Not a char
7530     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, ie, 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>.

```

7531     if dir == 'en' or dir == 'an' or dir == 'et' then
7532         if dir ~= 'et' then
7533             type_n = dir
7534         end
7535         first_n = first_n or item
7536         last_n = last_es or item
7537         last_es = nil
7538     elseif dir == 'es' and last_n then -- W3+W6
7539         last_es = item
7540     elseif dir == 'cs' then           -- it's right - do nothing
7541     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7542         if strong_lr == 'r' and type_n ~= '' then
7543             dir_mark(head, first_n, last_n, 'r')
7544         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7545             dir_mark(head, first_n, last_n, 'r')
7546             dir_mark(head, first_d, last_d, outer)
7547             first_d, last_d = nil, nil
7548         elseif strong_lr == 'l' and type_n ~= '' then
7549             last_d = last_n
7550         end
7551         type_n = ''
7552         first_n, last_n = nil, nil
7553     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:

```

7554     if dir == 'l' or dir == 'r' then
7555         if dir ~= outer then
7556             first_d = first_d or item
7557             last_d = item
7558         elseif first_d and dir ~= strong_lr then
7559             dir_mark(head, first_d, last_d, outer)
7560             first_d, last_d = nil, nil
7561         end
7562     end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resptly, but with other combinations depends on outer. From all

these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

7563   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7564       item.char = characters[item.char] and
7565           characters[item.char].m or item.char
7566   elseif (dir or new_dir) and last_lr ~= item then
7567       local mir = outer .. strong_lr .. (dir or outer)
7568       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7569           for ch in node.traverse(node.next(last_lr)) do
7570               if ch == item then break end
7571               if ch.id == node.id'glyph' and characters[ch.char] then
7572                   ch.char = characters[ch.char].m or ch.char
7573               end
7574           end
7575       end
7576   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).

```

7577   if dir == 'l' or dir == 'r' then
7578       last_lr = item
7579       strong = dir_real          -- Don't search back - best save now
7580       strong_lr = (strong == 'l') and 'l' or 'r'
7581   elseif new_dir then
7582       last_lr = nil
7583   end
7584 end

```

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

```

7585   if last_lr and outer == 'r' then
7586       for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7587           if characters[ch.char] then
7588               ch.char = characters[ch.char].m or ch.char
7589           end
7590       end
7591   end
7592   if first_n then
7593       dir_mark(head, first_n, last_n, outer)
7594   end
7595   if first_d then
7596       dir_mark(head, first_d, last_d, outer)
7597   end

```

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

```

7598   return node.prev(head) or head
7599 end
7600 </basic-r>

```

And here the Lua code for bidi=basic:

```

7601 (*basic)
7602 Babel = Babel or {}
7603
7604 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7605
7606 Babel.fontmap = Babel.fontmap or {}
7607 Babel.fontmap[0] = {}      -- l
7608 Babel.fontmap[1] = {}      -- r
7609 Babel.fontmap[2] = {}      -- al/an
7610
7611 -- To cancel mirroring. Also OML, OMS, U?
7612 Babel.symbol_fonts = Babel.symbol_fonts or {}

```



```

7613 Babel.symbol_fonts[font.id('tenln')] = true
7614 Babel.symbol_fonts[font.id('tenlnw')] = true
7615 Babel.symbol_fonts[font.id('tencirc')] = true
7616 Babel.symbol_fonts[font.id('tencircw')] = true
7617
7618 Babel.bidi_enabled = true
7619 Babel.mirroring_enabled = true
7620
7621 require('babel-data-bidi.lua')
7622
7623 local characters = Babel.characters
7624 local ranges = Babel.ranges
7625
7626 local DIR = node.id('dir')
7627 local GLYPH = node.id('glyph')
7628
7629 local function insert_implicit(head, state, outer)
7630   local new_state = state
7631   if state.sim and state.eim and state.sim ~= state.eim then
7632     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7633     local d = node.new(DIR)
7634     d.dir = '+' .. dir
7635     node.insert_before(head, state.sim, d)
7636     local d = node.new(DIR)
7637     d.dir = '-' .. dir
7638     node.insert_after(head, state.eim, d)
7639   end
7640   new_state.sim, new_state.eim = nil, nil
7641   return head, new_state
7642 end
7643
7644 local function insert_numeric(head, state)
7645   local new
7646   local new_state = state
7647   if state.san and state.ean and state.san ~= state.ean then
7648     local d = node.new(DIR)
7649     d.dir = '+TLT'
7650     _, new = node.insert_before(head, state.san, d)
7651     if state.san == state.sim then state.sim = new end
7652     local d = node.new(DIR)
7653     d.dir = '-TLT'
7654     _, new = node.insert_after(head, state.ean, d)
7655     if state.ean == state.eim then state.eim = new end
7656   end
7657   new_state.san, new_state.ean = nil, nil
7658   return head, new_state
7659 end
7660
7661 local function glyph_not_symbol_font(node)
7662   if node.id == GLYPH then
7663     return not Babel.symbol_fonts[node.font]
7664   else
7665     return false
7666   end
7667 end
7668
7669 -- TODO - \hbox with an explicit dir can lead to wrong results
7670 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7671 -- was s made to improve the situation, but the problem is the 3-dir
7672 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7673 -- well.
7674
7675 function Babel.bidi(head, ispar, hdir)

```

```

7676 local d    -- d is used mainly for computations in a loop
7677 local prev_d = ''
7678 local new_d = false
7679
7680 local nodes = {}
7681 local outer_first = nil
7682 local inmath = false
7683
7684 local glue_d = nil
7685 local glue_i = nil
7686
7687 local has_en = false
7688 local first_et = nil
7689
7690 local has_hyperlink = false
7691
7692 local ATDIR = Babel.attr_dir
7693
7694 local save_outer
7695 local temp = node.get_attribute(head, ATDIR)
7696 if temp then
7697     temp = temp & 0x3
7698     save_outer = (temp == 0 and 'l') or
7699                 (temp == 1 and 'r') or
7700                 (temp == 2 and 'al')
7701 elseif ispar then      -- Or error? Shouldn't happen
7702     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7703 else                  -- Or error? Shouldn't happen
7704     save_outer = ('TRT' == hdir) and 'r' or 'l'
7705 end
7706 -- when the callback is called, we are just _after_ the box,
7707 -- and the textdir is that of the surrounding text
7708 -- if not ispar and hdir ~= tex.textdir then
7709 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7710 -- end
7711 local outer = save_outer
7712 local last = outer
7713 -- 'al' is only taken into account in the first, current loop
7714 if save_outer == 'al' then save_outer = 'r' end
7715
7716 local fontmap = Babel.fontmap
7717
7718 for item in node.traverse(head) do
7719
7720     -- In what follows, #node is the last (previous) node, because the
7721     -- current one is not added until we start processing the neutrals.
7722
7723     -- three cases: glyph, dir, otherwise
7724     if glyph_not_symbol_font(item)
7725     or (item.id == 7 and item.subtype == 2) then
7726
7727         local d_font = nil
7728         local item_r
7729         if item.id == 7 and item.subtype == 2 then
7730             item_r = item.replace    -- automatic discs have just 1 glyph
7731         else
7732             item_r = item
7733         end
7734         local chardata = characters[item_r.char]
7735         d = chardata and chardata.d or nil
7736         if not d or d == 'nsm' then
7737             for nn, et in ipairs(ranges) do
7738                 if item_r.char < et[1] then

```

```

7739         break
7740     elseif item_r.char <= et[2] then
7741         if not d then d = et[3]
7742         elseif d == 'nsm' then d_font = et[3]
7743         end
7744         break
7745     end
7746 end
7747 end
7748 d = d or 'l'
7749
7750 -- A short 'pause' in bidi for mapfont
7751 d_font = d_font or d
7752 d_font = (d_font == 'l' and 0) or
7753           (d_font == 'nsm' and 0) or
7754           (d_font == 'r' and 1) or
7755           (d_font == 'al' and 2) or
7756           (d_font == 'an' and 2) or nil
7757 if d_font and fontmap and fontmap[d_font][item_r.font] then
7758     item_r.font = fontmap[d_font][item_r.font]
7759 end
7760
7761 if new_d then
7762     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7763     if inmath then
7764         attr_d = 0
7765     else
7766         attr_d = node.get_attribute(item, ATDIR)
7767         attr_d = attr_d & 0x3
7768     end
7769     if attr_d == 1 then
7770         outer_first = 'r'
7771         last = 'r'
7772     elseif attr_d == 2 then
7773         outer_first = 'r'
7774         last = 'al'
7775     else
7776         outer_first = 'l'
7777         last = 'l'
7778     end
7779     outer = last
7780     has_en = false
7781     first_et = nil
7782     new_d = false
7783 end
7784
7785 if glue_d then
7786     if (d == 'l' and 'l' or 'r') ~= glue_d then
7787         table.insert(nodes, {glue_i, 'on', nil})
7788     end
7789     glue_d = nil
7790     glue_i = nil
7791 end
7792
7793 elseif item.id == DIR then
7794     d = nil
7795
7796     if head ~= item then new_d = true end
7797
7798 elseif item.id == node.id'glue' and item.subtype == 13 then
7799     glue_d = d
7800     glue_i = item
7801     d = nil

```

```

7802
7803 elseif item.id == node.id'math' then
7804     inmath = (item.subtype == 0)
7805
7806 elseif item.id == 8 and item.subtype == 19 then
7807     has_hyperlink = true
7808
7809 else
7810     d = nil
7811 end
7812
7813 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7814 if last == 'al' and d == 'en' then
7815     d = 'an'           -- W3
7816 elseif last == 'al' and (d == 'et' or d == 'es') then
7817     d = 'on'           -- W6
7818 end
7819
7820 -- EN + CS/ES + EN      -- W4
7821 if d == 'en' and #nodes >= 2 then
7822     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7823         and nodes[#nodes-1][2] == 'en' then
7824         nodes[#nodes][2] = 'en'
7825     end
7826 end
7827
7828 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7829 if d == 'an' and #nodes >= 2 then
7830     if (nodes[#nodes][2] == 'cs')
7831         and nodes[#nodes-1][2] == 'an' then
7832         nodes[#nodes][2] = 'an'
7833     end
7834 end
7835
7836 -- ET/EN                -- W5 + W7->l / W6->on
7837 if d == 'et' then
7838     first_et = first_et or (#nodes + 1)
7839 elseif d == 'en' then
7840     has_en = true
7841     first_et = first_et or (#nodes + 1)
7842 elseif first_et then     -- d may be nil here !
7843     if has_en then
7844         if last == 'l' then
7845             temp = 'l'    -- W7
7846         else
7847             temp = 'en'   -- W5
7848         end
7849     else
7850         temp = 'on'       -- W6
7851     end
7852     for e = first_et, #nodes do
7853         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7854     end
7855     first_et = nil
7856     has_en = false
7857 end
7858
7859 -- Force mathdir in math if ON (currently works as expected only
7860 -- with 'l')
7861 if inmath and d == 'on' then
7862     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7863 end
7864

```

```

7865     if d then
7866         if d == 'al' then
7867             d = 'r'
7868             last = 'al'
7869         elseif d == 'l' or d == 'r' then
7870             last = d
7871         end
7872         prev_d = d
7873         table.insert(nodes, {item, d, outer_first})
7874     end
7875
7876     outer_first = nil
7877
7878 end
7879
7880 -- TODO -- repeated here in case EN/ET is the last node. Find a
7881 -- better way of doing things:
7882 if first_et then      -- dir may be nil here !
7883     if has_en then
7884         if last == 'l' then
7885             temp = 'l'      -- W7
7886         else
7887             temp = 'en'     -- W5
7888         end
7889     else
7890         temp = 'on'         -- W6
7891     end
7892     for e = first_et, #nodes do
7893         if glyph_not_symbol_font(nodes[e][1]) then nodes[e][2] = temp end
7894     end
7895 end
7896
7897 -- dummy node, to close things
7898 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7899
7900 ----- NEUTRAL -----
7901
7902 outer = save_outer
7903 last = outer
7904
7905 local first_on = nil
7906
7907 for q = 1, #nodes do
7908     local item
7909
7910     local outer_first = nodes[q][3]
7911     outer = outer_first or outer
7912     last = outer_first or last
7913
7914     local d = nodes[q][2]
7915     if d == 'an' or d == 'en' then d = 'r' end
7916     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7917
7918     if d == 'on' then
7919         first_on = first_on or q
7920     elseif first_on then
7921         if last == d then
7922             temp = d
7923         else
7924             temp = outer
7925         end
7926         for r = first_on, q - 1 do
7927             nodes[r][2] = temp

```

```

7928     item = nodes[r][1]    -- MIRRORING
7929     if Babel.mirroring_enabled and glyph_not_symbol_font(item)
7930         and temp == 'r' and characters[item.char] then
7931         local font_mode = ''
7932         if item.font > 0 and font.fonts[item.font].properties then
7933             font_mode = font.fonts[item.font].properties.mode
7934         end
7935         if font_mode ~= 'harf' and font_mode ~= 'plug' then
7936             item.char = characters[item.char].m or item.char
7937         end
7938     end
7939 end
7940 first_on = nil
7941 end
7942
7943 if d == 'r' or d == 'l' then last = d end
7944 end
7945
7946 ----- IMPLICIT, REORDER -----
7947
7948 outer = save_outer
7949 last = outer
7950
7951 local state = {}
7952 state.has_r = false
7953
7954 for q = 1, #nodes do
7955
7956     local item = nodes[q][1]
7957
7958     outer = nodes[q][3] or outer
7959
7960     local d = nodes[q][2]
7961
7962     if d == 'nsm' then d = last end          -- W1
7963     if d == 'en' then d = 'an' end
7964     local isdir = (d == 'r' or d == 'l')
7965
7966     if outer == 'l' and d == 'an' then
7967         state.san = state.san or item
7968         state.ean = item
7969     elseif state.san then
7970         head, state = insert_numeric(head, state)
7971     end
7972
7973     if outer == 'l' then
7974         if d == 'an' or d == 'r' then      -- im -> implicit
7975             if d == 'r' then state.has_r = true end
7976             state.sim = state.sim or item
7977             state.eim = item
7978         elseif d == 'l' and state.sim and state.has_r then
7979             head, state = insert_implicit(head, state, outer)
7980         elseif d == 'l' then
7981             state.sim, state.eim, state.has_r = nil, nil, false
7982         end
7983     else
7984         if d == 'an' or d == 'l' then
7985             if nodes[q][3] then -- nil except after an explicit dir
7986                 state.sim = item -- so we move sim 'inside' the group
7987             else
7988                 state.sim = state.sim or item
7989             end
7990             state.eim = item

```

```

7991     elseif d == 'r' and state.sim then
7992         head, state = insert_implicit(head, state, outer)
7993     elseif d == 'r' then
7994         state.sim, state.eim = nil, nil
7995     end
7996 end
7997
7998 if isdir then
7999     last = d          -- Don't search back - best save now
8000 elseif d == 'on' and state.san then
8001     state.san = state.san or item
8002     state.ean = item
8003 end
8004
8005 end
8006
8007 head = node.prev(head) or head
8008
8009 ----- FIX HYPERLINKS -----
8010
8011 if has_hyperlink then
8012     local flag, linking = 0, 0
8013     for item in node.traverse(head) do
8014         if item.id == DIR then
8015             if item.dir == '+TRT' or item.dir == '+TLT' then
8016                 flag = flag + 1
8017             elseif item.dir == '-TRT' or item.dir == '-TLT' then
8018                 flag = flag - 1
8019             end
8020             elseif item.id == 8 and item.subtype == 19 then
8021                 linking = flag
8022             elseif item.id == 8 and item.subtype == 20 then
8023                 if linking > 0 then
8024                     if item.prev.id == DIR and
8025                         (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
8026                         d = node.new(DIR)
8027                         d.dir = item.prev.dir
8028                         node.remove(head, item.prev)
8029                         node.insert_after(head, item, d)
8030                     end
8031                 end
8032                 linking = 0
8033             end
8034         end
8035     end
8036
8037     return head
8038 end
8039 </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.

```
8040 ⟨*nil⟩
8041 \ProvidesLanguage{nil}[\⟨date⟩ v\⟨version⟩ Nil language]
8042 \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.

```
8043 \ifx\l@nil\undefined
8044   \newlanguage\l@nil
8045   \@namedef{bbl@hyphendata@the\l@nil}{\{}}% Remove warning
8046   \let\bbl@elt\relax
8047   \edef\bbl@languages{% Add it to the list of languages
8048     \bbl@languages\bbl@elt{nil}{\the\l@nil}{\{}}
8049 \fi
```

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

```
8050 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
8051 \let\captionnil\@empty
8052 \let\datenil\@empty
```

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

```
8053 \def\bbl@inidata@nil{%
8054   \bbl@elt{identification}{tag.ini}{und}%
8055   \bbl@elt{identification}{load.level}{0}%
8056   \bbl@elt{identification}{charset}{utf8}%
8057   \bbl@elt{identification}{version}{1.0}%
8058   \bbl@elt{identification}{date}{2022-05-16}%
8059   \bbl@elt{identification}{name.local}{nil}%
8060   \bbl@elt{identification}{name.english}{nil}%
8061   \bbl@elt{identification}{name.babel}{nil}%
8062   \bbl@elt{identification}{tag.bcp47}{und}%
8063   \bbl@elt{identification}{language.tag.bcp47}{und}%
8064   \bbl@elt{identification}{tag.opentype}{dflt}%
8065   \bbl@elt{identification}{script.name}{Latin}%
8066   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
8067   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
8068   \bbl@elt{identification}{level}{1}%
8069   \bbl@elt{identification}{encodings}{}%
8070   \bbl@elt{identification}{derive}{no}}
8071 \@namedef{bbl@tbc@nil}{und}
8072 \@namedef{bbl@lbc@nil}{und}
8073 \@namedef{bbl@casing@nil}{und} % TODO
8074 \@namedef{bbl@lotf@nil}{dflt}
8075 \@namedef{bbl@elname@nil}{nil}
8076 \@namedef{bbl@lname@nil}{nil}
8077 \@namedef{bbl@esname@nil}{Latin}
8078 \@namedef{bbl@sname@nil}{Latin}
8079 \@namedef{bbl@sbc@nil}{Latn}
8080 \@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.

```
8081 \ldf@finish{nil}
8082 ⟨/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.

```
8083 <<Compute Julian day>> ≡
8084 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
8085 \def\bbl@cs@gregleap#1{%
8086   (\bbl@fpmo{#1}{4} == 0) &&
8087   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
8088 \def\bbl@cs@jd#1#2#3{% year, month, day
8089   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
8090     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
8091     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
8092     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3)} }
8093 <</Compute Julian day>>
```

13.1 Islamic

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

```
8094 <*ca-islamic>
8095 \ExplSyntaxOn
8096 <<Compute Julian day>>
8097 % == islamic (default)
8098 % Not yet implemented
8099 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar:

```
8100 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
8101   ((#3 + ceil(29.5 * (#2 - 1)) +
8102     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
8103     1948439.5) - 1) }
8104 \@namedef{\bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
8105 \@namedef{\bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
8106 \@namedef{\bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
8107 \@namedef{\bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
8108 \@namedef{\bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
8109 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
8110   \edef\bbl@tempa{%
8111     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
8112   \edef#5{%
8113     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
8114   \edef#6{\fp_eval:n{
8115     min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
8116   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}
```

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

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

```
8117 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
8118 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
8119 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
8120 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
8121 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
8122 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
8123 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
8124 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
8125 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
8126 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
8127 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
8128 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
```

```

8129 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
8130 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
8131 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
8132 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
8133 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
8134 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
8135 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
8136 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
8137 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
8138 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
8139 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
8140 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
8141 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
8142 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
8143 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
8144 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
8145 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
8146 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
8147 65401,65431,65460,65490,65520}
8148 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
8149 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
8150 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
8151 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
8152   \ifnum#2>2014 \ifnum#2<2038
8153     \bbl@afterfi\expandafter\@gobble
8154   \fi\fi
8155   {\bbl@error{year-out-range}{2014-2038}{}}}%
8156 \def\bbl@tempd{\fp_eval:n{ % (Julian) day
8157   \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
8158 \count@\@ne
8159 \bbl@foreach\bbl@cs@umalqura@data{%
8160   \advance\count@\@ne
8161   \ifnum##1>\bbl@tempd\else
8162     \edef\bbl@tempe{\the\count@}%
8163     \edef\bbl@tempb{##1}%
8164   \fi}%
8165 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8166 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1) / 12) }}% annus
8167 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8168 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8169 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8170 \ExplSyntaxOff
8171 \bbl@add\bbl@precalendar{%
8172   \bbl@replace\bbl@ld@calendar{-civil}{}}%
8173   \bbl@replace\bbl@ld@calendar{-umalqura}{}}%
8174   \bbl@replace\bbl@ld@calendar{+}{}}%
8175   \bbl@replace\bbl@ld@calendar{-}{}}%
8176 \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`

```

8177 (*ca-hebrew)
8178 \newcount\bbl@cntcommon
8179 \def\bbl@remainder#1#2#3{%
8180   #3=#1\relax
8181   \divide #3 by #2\relax
8182   \multiply #3 by -#2\relax
8183   \advance #3 by #1\relax}%
8184 \newif\ifbbl@divisible
8185 \def\bbl@checkifdivisible#1#2{%

```

```

8186 {\countdef\tmp=0
8187 \bbl@remainder{#1}{#2}{\tmp}%
8188 \ifnum \tmp=0
8189 \global\bbl@divisibletrue
8190 \else
8191 \global\bbl@divisiblefalse
8192 \fi}}
8193 \newif\ifbbl@gregleap
8194 \def\bbl@ifgregleap#1{%
8195 \bbl@checkifdivisible{#1}{4}%
8196 \ifbbl@divisible
8197 \bbl@checkifdivisible{#1}{100}%
8198 \ifbbl@divisible
8199 \bbl@checkifdivisible{#1}{400}%
8200 \ifbbl@divisible
8201 \bbl@gregleaptrue
8202 \else
8203 \bbl@gregleapfalse
8204 \fi
8205 \else
8206 \bbl@gregleaptrue
8207 \fi
8208 \else
8209 \bbl@gregleapfalse
8210 \fi
8211 \ifbbl@gregleap}
8212 \def\bbl@gregdayspriormonths#1#2#3{%
8213 {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8214 181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8215 \bbl@ifgregleap{#2}%
8216 \ifnum #1 > 2
8217 \advance #3 by 1
8218 \fi
8219 \fi
8220 \global\bbl@cntcommon=#3}%
8221 #3=\bbl@cntcommon}
8222 \def\bbl@gregdaysprioryears#1#2{%
8223 {\countdef\tmpc=4
8224 \countdef\tmpb=2
8225 \tmpb=#1\relax
8226 \advance \tmpb by -1
8227 \tmpc=\tmpb
8228 \multiply \tmpc by 365
8229 #2=\tmpc
8230 \tmpc=\tmpb
8231 \divide \tmpc by 4
8232 \advance #2 by \tmpc
8233 \tmpc=\tmpb
8234 \divide \tmpc by 100
8235 \advance #2 by -\tmpc
8236 \tmpc=\tmpb
8237 \divide \tmpc by 400
8238 \advance #2 by \tmpc
8239 \global\bbl@cntcommon=#2\relax}%
8240 #2=\bbl@cntcommon}
8241 \def\bbl@absfromgreg#1#2#3#4{%
8242 {\countdef\tmpd=0
8243 #4=#1\relax
8244 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8245 \advance #4 by \tmpd
8246 \bbl@gregdaysprioryears{#3}{\tmpd}%
8247 \advance #4 by \tmpd
8248 \global\bbl@cntcommon=#4\relax}%

```

```

8249 #4=\bbl@cntcommon}
8250 \newif\ifbbl@hebrleap
8251 \def\bbl@checkleaphebrewyear#1{%
8252   {\countdef\tmpa=0
8253    \countdef\tmpb=1
8254    \tmpa=#1\relax
8255    \multiply \tmpa by 7
8256    \advance \tmpa by 1
8257    \bbl@remainder{\tmpa}{19}{\tmpb}%
8258    \ifnum \tmpb < 7
8259      \global\bbl@hebrleaptrue
8260    \else
8261      \global\bbl@hebrleapfalse
8262    \fi}}
8263 \def\bbl@hebreleapsedmonths#1#2{%
8264   {\countdef\tmpa=0
8265    \countdef\tmpb=1
8266    \countdef\tmpc=2
8267    \tmpa=#1\relax
8268    \advance \tmpa by -1
8269    #2=\tmpa
8270    \divide #2 by 19
8271    \multiply #2 by 235
8272    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8273    \tmpc=\tmpb
8274    \multiply \tmpb by 12
8275    \advance #2 by \tmpb
8276    \multiply \tmpc by 7
8277    \advance \tmpc by 1
8278    \divide \tmpc by 19
8279    \advance #2 by \tmpc
8280    \global\bbl@cntcommon=#2}%
8281 #2=\bbl@cntcommon}
8282 \def\bbl@hebreleapseddays#1#2{%
8283   {\countdef\tmpa=0
8284    \countdef\tmpb=1
8285    \countdef\tmpc=2
8286    \bbl@hebreleapsedmonths{#1}{#2}%
8287    \tmpa=#2\relax
8288    \multiply \tmpa by 13753
8289    \advance \tmpa by 5604
8290    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8291    \divide \tmpa by 25920
8292    \multiply #2 by 29
8293    \advance #2 by 1
8294    \advance #2 by \tmpa
8295    \bbl@remainder{#2}{7}{\tmpa}%
8296    \ifnum \tmpc < 19440
8297      \ifnum \tmpc < 9924
8298      \else
8299        \ifnum \tmpa=2
8300          \bbl@checkleaphebrewyear{#1}% of a common year
8301          \ifbbl@hebrleap
8302          \else
8303            \advance #2 by 1
8304          \fi
8305        \fi
8306      \fi
8307      \ifnum \tmpc < 16789
8308      \else
8309        \ifnum \tmpa=1
8310          \advance #1 by -1
8311          \bbl@checkleaphebrewyear{#1}% at the end of leap year

```

```

8312             \ifbbl@hebrleap
8313             \advance #2 by 1
8314         \fi
8315     \fi
8316 \fi
8317 \else
8318     \advance #2 by 1
8319 \fi
8320 \bbl@remainder{#2}{7}{\tmpa}%
8321 \ifnum \tmpa=0
8322     \advance #2 by 1
8323 \else
8324     \ifnum \tmpa=3
8325         \advance #2 by 1
8326     \else
8327         \ifnum \tmpa=5
8328             \advance #2 by 1
8329         \fi
8330     \fi
8331 \fi
8332 \global\bbl@cntcommon=#2\relax}%
8333 #2=\bbl@cntcommon}
8334 \def\bbl@daysinhebryear#1#2{%
8335     {\countdef\tmpe=12
8336     \bbl@hebreleapseddays{#1}{\tmpe}%
8337     \advance #1 by 1
8338     \bbl@hebreleapseddays{#1}{#2}%
8339     \advance #2 by -\tmpe
8340     \global\bbl@cntcommon=#2}%
8341     #2=\bbl@cntcommon}
8342 \def\bbl@hebrdayspriormonths#1#2#3{%
8343     {\countdef\tmpf= 14
8344     #3=\ifcase #1\relax
8345         0 \or
8346         0 \or
8347         30 \or
8348         59 \or
8349         89 \or
8350         118 \or
8351         148 \or
8352         148 \or
8353         177 \or
8354         207 \or
8355         236 \or
8356         266 \or
8357         295 \or
8358         325 \or
8359         400
8360     \fi
8361     \bbl@checkleaphebryear{#2}%
8362     \ifbbl@hebrleap
8363         \ifnum #1 > 6
8364             \advance #3 by 30
8365         \fi
8366     \fi
8367     \bbl@daysinhebryear{#2}{\tmpf}%
8368     \ifnum #1 > 3
8369         \ifnum \tmpf=353
8370             \advance #3 by -1
8371         \fi
8372         \ifnum \tmpf=383
8373             \advance #3 by -1
8374         \fi

```

```

8375 \fi
8376 \ifnum #1 > 2
8377     \ifnum \tmpf=355
8378         \advance #3 by 1
8379     \fi
8380     \ifnum \tmpf=385
8381         \advance #3 by 1
8382     \fi
8383 \fi
8384 \global\bbbl@cntcommon=#3\relax}%
8385 #3=\bbbl@cntcommon}
8386 \def\bbbl@absfromhebr#1#2#3#4{%
8387     {#4=#1\relax
8388     \bbbl@hebrdayspriormonths{#2}{#3}{#1}%
8389     \advance #4 by #1\relax
8390     \bbbl@hebrrelapseddays{#3}{#1}%
8391     \advance #4 by #1\relax
8392     \advance #4 by -1373429
8393     \global\bbbl@cntcommon=#4\relax}%
8394 #4=\bbbl@cntcommon}
8395 \def\bbbl@hebrfromgreg#1#2#3#4#5#6{%
8396     {\countdef\tmpx= 17
8397     \countdef\tmpy= 18
8398     \countdef\tmpz= 19
8399     #6=#3\relax
8400     \global\advance #6 by 3761
8401     \bbbl@absfromgreg{#1}{#2}{#3}{#4}%
8402     \tmpz=1 \tmpy=1
8403     \bbbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8404     \ifnum \tmpx > #4\relax
8405         \global\advance #6 by -1
8406         \bbbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8407     \fi
8408     \advance #4 by -\tmpx
8409     \advance #4 by 1
8410     #5=#4\relax
8411     \divide #5 by 30
8412     \loop
8413         \bbbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8414         \ifnum \tmpx < #4\relax
8415             \advance #5 by 1
8416             \tmpy=\tmpx
8417         \repeat
8418     \global\advance #5 by -1
8419     \global\advance #4 by -\tmpy}}
8420 \newcount\bbbl@hebrday \newcount\bbbl@hebrmonth \newcount\bbbl@hebryear
8421 \newcount\bbbl@gregday \newcount\bbbl@gregmonth \newcount\bbbl@gregyear
8422 \def\bbbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8423     \bbbl@gregday=#3\relax \bbbl@gregmonth=#2\relax \bbbl@gregyear=#1\relax
8424     \bbbl@hebrfromgreg
8425         {\bbbl@gregday}{\bbbl@gregmonth}{\bbbl@gregyear}%
8426     {\bbbl@hebrday}{\bbbl@hebrmonth}{\bbbl@hebryear}%
8427     \edef#4{\the\bbbl@hebryear}%
8428     \edef#5{\the\bbbl@hebrmonth}%
8429     \edef#6{\the\bbbl@hebrday}}
8430 </ca-hebrew>

```

13.3 Persian

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

pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```

8431 (*ca-persian)
8432 \ExplSyntaxOn
8433 <<Compute Julian day>>
8434 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8435   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8436 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8437   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8438   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8439     \bbl@afterfi\expandafter@gobble
8440   \fi\fi
8441   {\bbl@error{year-out-range}{2013-2050}{}}}%
8442   \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8443   \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8444   \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8445   \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8446   \ifnum\bbl@tempc<\bbl@tempb
8447     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8448     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8449     \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8450     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8451   \fi
8452   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8453   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8454   \edef#5{\fp_eval:n{% set Jalali month
8455     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8456   \edef#6{\fp_eval:n{% set Jalali day
8457     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8458 \ExplSyntaxOff
8459 </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.

```

8460 (*ca-coptic)
8461 \ExplSyntaxOn
8462 <<Compute Julian day>>
8463 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8464   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8465   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8466   \edef#4{\fp_eval:n{%
8467     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8468   \edef\bbl@tempc{\fp_eval:n{%
8469     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8470   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8471   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8472 \ExplSyntaxOff
8473 </ca-coptic>
8474 (*ca-ethiopic)
8475 \ExplSyntaxOn
8476 <<Compute Julian day>>
8477 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8478   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8479   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8480   \edef#4{\fp_eval:n{%
8481     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8482   \edef\bbl@tempc{\fp_eval:n{%
8483     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8484   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8485   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8486 \ExplSyntaxOff

```

8487 </ca-ethiopic>

13.5 Buddhist

That's very simple.

```
8488 <*ca-buddhist>
8489 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8490   \edef#4{\number\numexpr#1+543\relax}%
8491   \edef#5{#2}%
8492   \edef#6{#3}}
8493 </ca-buddhist>
8494 %
8495 % \subsection{Chinese}
8496 %
8497 % Brute force, with the Julian day of first day of each month. The
8498 % table has been computed with the help of \textsf{python-lunardate} by
8499 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8500 % is 2015-2044.
8501 %
8502 % \begin{macrocode}
8503 <*ca-chinese>
8504 \ExplSyntaxOn
8505 <<Compute Julian day>>
8506 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8507   \edef\bbl@tempd{\fp_eval:n{%
8508     \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8509   \count@ \z@
8510   \@tempcnta=2015
8511   \bbl@foreach\bbl@cs@chinese@data{%
8512     \ifnum##1>\bbl@tempd\else
8513       \advance\count@\@ne
8514       \ifnum\count@>12
8515         \count@\@ne
8516         \advance\@tempcnta\@ne\fi
8517       \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8518       \ifin@
8519         \advance\count@\m@ne
8520       \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8521       \else
8522         \edef\bbl@tempe{\the\count@}%
8523       \fi
8524       \edef\bbl@tempb{##1}%
8525       \fi}%
8526   \edef#4{\the\@tempcnta}%
8527   \edef#5{\bbl@tempe}%
8528   \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8529 \def\bbl@cs@chinese@leap{%
8530   885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8531 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8532   354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8533   768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8534   1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8535   1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8536   1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8537   2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8538   2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8539   2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8540   3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8541   3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8542   3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8543   4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8544   4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8545   5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
```



```

8546 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8547 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8548 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8549 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8550 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8551 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8552 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8553 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8554 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8555 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8556 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8557 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8558 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8559 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8560 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8561 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8562 10896,10926,10956,10986,11015,11045,11074,11103}
8563 \ExplSyntaxOff
8564 </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`.

```

8565 (*bplain | blplain)
8566 \catcode\{=1 % left brace is begin-group character
8567 \catcode\}=2 % right brace is end-group character
8568 \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).

```

8569 \openin 0 hyphen.cfg
8570 \ifeof0
8571 \else
8572 \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.

```

8573 \def\input #1 {%
8574 \let\input\input
8575 \a hyphen.cfg
8576 \let\input\undefined
8577 }
8578 \fi
8579 </bplain | blplain>

```

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

```
8580 \bplain\la plain.tex
8581 \bplain\la 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.

```
8582 \bplain\def\fmtname{babel-plain}
8583 \bplain\def\fmtname{babel-lplain}
```

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

14.2 Emulating some \LaTeX features

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

```
8584 \langle\langle *Emulate LaTeX \rangle\rangle \equiv
8585 \def\@empty{}
8586 \def\loadlocalcfg#1{%
8587   \openin0#1.cfg
8588   \ifeof0
8589     \closein0
8590   \else
8591     \closein0
8592     {\immediate\writel6{*****}%
8593      \immediate\writel6{* Local config file #1.cfg used}%
8594      \immediate\writel6{*}%
8595     }
8596     \input #1.cfg\relax
8597   \fi
8598   \@endoflfd}
```

14.3 General tools

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

```
8599 \long\def\@firstofone#1{#1}
8600 \long\def\@firstoftwo#1#2{#1}
8601 \long\def\@secondoftwo#1#2{#2}
8602 \def\@nnil{\@nil}
8603 \def\@gobbletwo#1#2{}
8604 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8605 \def\@star@or@long#1{%
8606   \@ifstar
8607   {\let\l@ngrel@x\relax#1}%
8608   {\let\l@ngrel@x\long#1}}
8609 \let\l@ngrel@x\relax
8610 \def\@car#1#2\@nil{#1}
8611 \def\@cdr#1#2\@nil{#2}
8612 \let\@typeset@protect\relax
8613 \let\protected@edef\edef
8614 \long\def\@gobble#1{}
8615 \edef\@backslashchar{\expandafter\@gobble\string\}
8616 \def\strip@prefix#1>{}
8617 \def\g@addto@macro#1#2{{%
8618   \toks@{\expandafter{#1#2}}%
8619   \xdef#1{\the\toks@}}}
8620 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8621 \def\@nameuse#1{\csname #1\endcsname}
```

```

8622 \def\@ifundefined#1{%
8623   \expandafter\ifx\csname#1\endcsname\relax
8624     \expandafter\@firstoftwo
8625   \else
8626     \expandafter\@secondoftwo
8627   \fi}
8628 \def\@expandtwoargs#1#2#3{%
8629   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8630 \def\zap@space#1 #2{%
8631   #1%
8632   \ifx#2\@empty\else\expandafter\zap@space\fi
8633   #2}
8634 \let\bbl@trace\@gobble
8635 \def\bbl@error#1{% Implicit #2#3#4
8636   \begingroup
8637     \catcode`\=0 \catcode`\==12 \catcode`\`=12
8638     \catcode`\^M=5 \catcode`\%=14
8639     \input errbabel.def
8640   \endgroup
8641   \bbl@error{#1}}
8642 \def\bbl@warning#1{%
8643   \begingroup
8644     \newlinechar=`^^J
8645     \def\{^^J(babel) }%
8646     \message{\#1}%
8647   \endgroup}
8648 \let\bbl@infowarn\bbl@warning
8649 \def\bbl@info#1{%
8650   \begingroup
8651     \newlinechar=`^^J
8652     \def\{^^J}%
8653     \wlog{#1}%
8654   \endgroup}

```

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

```

8655 \ifx\@preamblecmds\@undefined
8656   \def\@preamblecmds{}
8657 \fi
8658 \def\@onlypreamble#1{%
8659   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8660     \@preamblecmds\do#1}}
8661 \@onlypreamble\@onlypreamble

```

Mimic \LaTeX 's `\AtBeginDocument`; for this to work the user needs to add `\begin{document}` to his file.

```

8662 \def\begin{document}{%
8663   \@begin{document}hook
8664   \global\let\@begin{document}hook\@undefined
8665   \def\do##1{\global\let##1\@undefined}%
8666   \@preamblecmds
8667   \global\let\do\noexpand}

8668 \ifx\@begin{document}hook\@undefined
8669   \def\@begin{document}hook{}
8670 \fi
8671 \@onlypreamble\@begin{document}hook
8672 \def\AtBeginDocument{\g@addto@macro\@begin{document}hook}

```

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

```

8673 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8674 \@onlypreamble\AtEndOfPackage
8675 \def\@endofldf{}
8676 \@onlypreamble\@endofldf

```

```

8677 \let\bbl@afterlang\@empty
8678 \chardef\bbl@opt@hyphenmap\z@

```

\TeX 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.

```

8679 \catcode`\&=\z@
8680 \ifx&\if@files\@undefined
8681   \expandafter\let\csname if@files\expandafter\endcsname
8682     \csname iffalse\endcsname
8683 \fi
8684 \catcode`\&=4

```

Mimic \TeX 's commands to define control sequences.

```

8685 \def\newcommand{\@star@or@long\new@command}
8686 \def\new@command#1{%
8687   \@testopt{\@newcommand#1}0}
8688 \def\@newcommand#1[#2]{%
8689   \@ifnextchar [{\@xargdef#1[#2]}%
8690     {\@argdef#1[#2]}}
8691 \long\def\@argdef#1[#2]#3{%
8692   \@yargdef#1\@ne{#2}{#3}}
8693 \long\def\@xargdef#1[#2][#3]#4{%
8694   \expandafter\def\expandafter#1\expandafter{%
8695     \expandafter\@protected@testopt\expandafter #1%
8696     \csname\string#1\expandafter\endcsname{#3}}%
8697   \expandafter\@yargdef \csname\string#1\endcsname
8698   \tw@{#2}{#4}}
8699 \long\def\@yargdef#1#2#3{%
8700   \@tempcnta#3\relax
8701   \advance \@tempcnta \@ne
8702   \let\@hash@\relax
8703   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8704   \@tempcntb #2%
8705   \@whilenum\@tempcntb <\@tempcnta
8706   \do{%
8707     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8708     \advance\@tempcntb \@ne}%
8709   \let\@hash@##%
8710   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8711 \def\providecommand{\@star@or@long\provide@command}
8712 \def\provide@command#1{%
8713   \begingroup
8714     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8715   \endgroup
8716   \expandafter\@ifundefined\@gtempa
8717     {\def\reserved@a{\new@command#1}}%
8718     {\let\reserved@a\relax
8719     \def\reserved@a{\new@command\reserved@a}}%
8720   \reserved@a}%
8721 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8722 \def\declare@robustcommand#1{%
8723   \edef\reserved@a{\string#1}%
8724   \def\reserved@b{#1}%
8725   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8726   \edef#1{%
8727     \ifx\reserved@a\reserved@b
8728       \noexpand\x@protect
8729       \noexpand#1%
8730     \fi
8731     \noexpand\protect
8732     \expandafter\@gobble\string#1 \endcsname
8733   }

```

```

8734 }%
8735 \expandafter\newcommand\csname
8736 \expandafter\@gobble\string#1 \endcsname
8737 }
8738 \def\x@protect#1{%
8739 \ifx\protect\@typeset@protect\else
8740 \x@protect#1%
8741 \fi
8742 }
8743 \catcode`\&=\z@ % Trick to hide conditionals
8744 \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`.

```

8745 \def\bbl@tempa{\csname newif\endcsname&fin@}
8746 \catcode`\&=4
8747 \ifx\in@\undefined
8748 \def\in@#1#2{%
8749 \def\in@@##1#1##2##3\in@@{%
8750 \ifx\in@##2\in@false\else\in@true\fi}%
8751 \in@@#2#1\in@\in@@}
8752 \else
8753 \let\bbl@tempa\@empty
8754 \fi
8755 \bbl@tempa

```

\LaTeX 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).

```

8756 \def\@ifpackagewith#1#2#3#4{#3}

```

The \LaTeX 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.

```

8757 \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 $\LaTeX 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

8758 \ifx\@tempcnta\undefined
8759 \csname newcount\endcsname\@tempcnta\relax
8760 \fi
8761 \ifx\@tempcntb\undefined
8762 \csname newcount\endcsname\@tempcntb\relax
8763 \fi

```

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

```

8764 \ifx\bye\undefined
8765 \advance\count10 by -2\relax
8766 \fi
8767 \ifx\@ifnextchar\undefined
8768 \def\@ifnextchar#1#2#3{%
8769 \let\reserved@d=#1%
8770 \def\reserved@a{#2}\def\reserved@b{#3}%
8771 \futurelet\@let@token\@ifnch}
8772 \def\@ifnch{%
8773 \ifx\@let@token\@sptoken
8774 \let\reserved@c\@xifnch
8775 \else
8776 \ifx\@let@token\reserved@d
8777 \let\reserved@c\reserved@a

```

```

8778     \else
8779         \let\reserved@c\reserved@b
8780     \fi
8781 \fi
8782 \reserved@c}
8783 \def\:{\let\sptoken= } \: % this makes \@sptoken a space token
8784 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
8785 \fi
8786 \def\@testopt#1#2{%
8787     \@ifnextchar[{\#1}{\#1[\#2]}}
8788 \def\@protected@testopt#1{%
8789     \ifx\protect\@typeset@protect
8790         \expandafter\@testopt
8791     \else
8792         \@x@protect#1%
8793     \fi}
8794 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8795     #2\relax}\fi}
8796 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8797     \else\expandafter\@gobble\fi{#1}}

```

14.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ environment.

```

8798 \def\DeclareTextCommand{%
8799     \@dec@text@cmd\providecommand
8800 }
8801 \def\ProvideTextCommand{%
8802     \@dec@text@cmd\providecommand
8803 }
8804 \def\DeclareTextSymbol#1#2#3{%
8805     \@dec@text@cmd\chardef#1{\#2}\#3\relax
8806 }
8807 \def\@dec@text@cmd#1#2#3{%
8808     \expandafter\def\expandafter#2%
8809         \expandafter{%
8810             \csname#3-cmd\expandafter\endcsname
8811             \expandafter#2%
8812             \csname#3\string#2\endcsname
8813         }%
8814 % \let\@ifdefinable\@rc@ifdefinable
8815 \expandafter#1\csname#3\string#2\endcsname
8816 }
8817 \def\@current@cmd#1{%
8818     \ifx\protect\@typeset@protect\else
8819         \noexpand#1\expandafter\@gobble
8820     \fi
8821 }
8822 \def\@changed@cmd#1#2{%
8823     \ifx\protect\@typeset@protect
8824         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8825             \expandafter\ifx\csname ?\string#1\endcsname\relax
8826                 \expandafter\def\csname ?\string#1\endcsname{%
8827                     \@changed@x@err{#1}%
8828                 }%
8829             \fi
8830             \global\expandafter\let
8831                 \csname\cf@encoding\string#1\expandafter\endcsname
8832                 \csname ?\string#1\endcsname
8833             \fi
8834             \csname\cf@encoding\string#1%
8835                 \expandafter\endcsname
8836     \else

```

```

8837     \noexpand#1%
8838     \fi
8839 }
8840 \def\@changed@x@err#1{%
8841     \errhelp{Your command will be ignored, type <return> to proceed}%
8842     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8843 \def\DeclareTextCommandDefault#1{%
8844     \DeclareTextCommand#1?%
8845 }
8846 \def\ProvideTextCommandDefault#1{%
8847     \ProvideTextCommand#1?%
8848 }
8849 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8850 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8851 \def\DeclareTextAccent#1#2#3{%
8852     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8853 }
8854 \def\DeclareTextCompositeCommand#1#2#3#4{%
8855     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8856     \edef\reserved@b{\string##1}%
8857     \edef\reserved@c{%
8858         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8859     \ifx\reserved@b\reserved@c
8860         \expandafter\expandafter\expandafter\ifx
8861             \expandafter\@car\reserved@a\relax\relax\@nil
8862             \@text@composite
8863         \else
8864             \edef\reserved@b##1{%
8865                 \def\expandafter\noexpand
8866                     \csname#2\string#1\endcsname####1{%
8867                     \noexpand\@text@composite
8868                         \expandafter\noexpand\csname#2\string#1\endcsname
8869                         ####1\noexpand\@empty\noexpand\@text@composite
8870                         {##1}%
8871                     }%
8872                 }%
8873             \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8874         \fi
8875         \expandafter\def\csname\expandafter\string\csname
8876             #2\endcsname\string#1-\string#3\endcsname{#4}
8877     \else
8878         \errhelp{Your command will be ignored, type <return> to proceed}%
8879         \errmessage{\string\DeclareTextCompositeCommand\space used on
8880             inappropriate command \protect#1}
8881     \fi
8882 }
8883 \def\@text@composite#1#2#3\@text@composite{%
8884     \expandafter\@text@composite@x
8885         \csname\string#1-\string#2\endcsname
8886 }
8887 \def\@text@composite@x#1#2{%
8888     \ifx#1\relax
8889         #2%
8890     \else
8891         #1%
8892     \fi
8893 }
8894 %
8895 \def\@strip@args#1:#2-#3\@strip@args{#2}
8896 \def\DeclareTextComposite#1#2#3#4{%
8897     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8898     \bgroup
8899     \lccode`\@=#4%

```

```

8900     \lowercase{%
8901     \egroup
8902     \reserved@a @%
8903     }%
8904 }
8905 %
8906 \def\UseTextSymbol#1#2{#2}
8907 \def\UseTextAccent#1#2#3{}
8908 \def\@use@text@encoding#1{}
8909 \def\DeclareTextSymbolDefault#1#2{%
8910     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8911 }
8912 \def\DeclareTextAccentDefault#1#2{%
8913     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8914 }
8915 \def\cf@encoding{OT1}

```

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

```

8916 \DeclareTextAccent{"}{OT1}{127}
8917 \DeclareTextAccent{'}{OT1}{19}
8918 \DeclareTextAccent{^}{OT1}{94}
8919 \DeclareTextAccent{`}{OT1}{18}
8920 \DeclareTextAccent{~}{OT1}{126}

```

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

```

8921 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8922 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
8923 \DeclareTextSymbol{\textquoteleft}{OT1}{`'}
8924 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
8925 \DeclareTextSymbol{\i}{OT1}{16}
8926 \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`.

```

8927 \ifx\scriptsize\@undefined
8928     \let\scriptsize\sevenrm
8929 \fi

```

And a few more “dummy” definitions.

```

8930 \def\language{english}%
8931 \let\bbl@opt@shorthands\@nnil
8932 \def\bbl@ifshorthand#1#2#3{#2}%
8933 \let\bbl@language@opts\@empty
8934 \let\bbl@ensureinfo\@gobble
8935 \let\bbl@provide@locale\relax
8936 \ifx\babeloptionstrings\@undefined
8937     \let\bbl@opt@strings\@nnil
8938 \else
8939     \let\bbl@opt@strings\babeloptionstrings
8940 \fi
8941 \def\BabelStringsDefault{generic}
8942 \def\bbl@tempa{normal}
8943 \ifx\babeloptionmath\bbl@tempa
8944     \def\bbl@mathnormal{\noexpand\textormath}
8945 \fi
8946 \def\AfterBabelLanguage#1#2{}
8947 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8948 \let\bbl@afterlang\relax
8949 \def\bbl@opt@safe{BR}
8950 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8951 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8952 \expandafter\newif\csname ifbbl@single\endcsname
8953 \chardef\bbl@bidimode\z@
8954 <</Emulate LaTeX>>

```


A proxy file:

```
8955 <*plain>
8956 \input babel.def
8957 </plain>
```

15 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

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

References

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