

Babel

Code

Version 3.89.17574
2023/06/24

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 (<code>\LaTeX</code> , <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	40
4.8	Short tags	41
4.9	Hyphens	42
4.10	Multiencoding strings	43
4.11	Macros common to a number of languages	49
4.12	Making glyphs available	49
4.12.1	Quotation marks	49
4.12.2	Letters	51
4.12.3	Shorthands for quotation marks	52
4.12.4	Umlauts and tremas	52
4.13	Layout	54
4.14	Load engine specific macros	54
4.15	Creating and modifying languages	55
5	Adjusting the Babel bahavior	77
5.1	Cross referencing macros	79
5.2	Marks	82
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>	84
5.4	Encoding and fonts	84
5.5	Basic bidi support	86
5.6	Local Language Configuration	89
5.7	Language options	89
6	The kernel of Babel (<code>babel.def</code>, <code>common</code>)	93
7	Loading hyphenation patterns	93
8	Font handling with <code>fontspec</code>	97
9	Hooks for XeTeX and LuaTeX	100
9.1	XeTeX	100
9.2	Layout	102
9.3	8-bit TeX	104
9.4	LuaTeX	104
9.5	Southeast Asian scripts	110
9.6	CJK line breaking	112

9.7	Arabic justification	114
9.8	Common stuff	118
9.9	Automatic fonts and ids switching	118
9.10	Bidi	124
9.11	Layout	126
9.12	Lua: transforms	133
9.13	Lua: Auto bidi with basic and basic-r	141
10	Data for CJK	152
11	The ‘nil’ language	152
12	Calendars	153
12.1	Islamic	154
12.2	Hebrew	155
12.3	Persian	159
12.4	Coptic and Ethiopic	160
12.5	Buddhist	160
13	Support for Plain T_EX (plain.def)	160
13.1	Not renaming hyphen.tex	160
13.2	Emulating some L ^A T _E X features	161
13.3	General tools	162
13.4	Encoding related macros	165
14	Acknowledgements	168

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 appropriated 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=3.89.17574>>
2 <<date=2023/06/24>>
```

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

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

```
3 <<*Basic macros>> ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1#2}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@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\bb@ifunset#1{%
58     \expandafter\ifx\csname#1\endcsname\relax
59       \expandafter\@firstoftwo
60     \else
61       \expandafter\@secondoftwo
62     \fi}
63 \bb@ifunset{ifcsname}%
64 {}%
65 {\gdef\bb@ifunset#1{%
66   \ifcsname#1\endcsname
67     \expandafter\ifx\csname#1\endcsname\relax
68       \bb@afterelse\expandafter\@firstoftwo
69     \else
70       \bb@afterfi\expandafter\@secondoftwo
71     \fi
72   \else
73     \expandafter\@firstoftwo
74   \fi}}
75 \endgroup

```

`\bb@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\bb@ifblank#1{%
77   \bb@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bb@ifblank@i#1#2\@nil#3#4#5\@nil#4}
79 \def\bb@ifset#1#2#3{%
80   \bb@ifunset{#1}{#3}{\bb@exp{\bb@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\bb@forkv#1#2{%
82   \def\bb@kvcmd##1##2##3{#2}%
83   \bb@kvnext#1,\@nil,}
84 \def\bb@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86     \bb@ifblank{#1}{\bb@forkv@eq#1=\@empty=\@nil{#1}}%
87     \expandafter\bb@kvnext
88   \fi}
89 \def\bb@forkv@eq#1=#2=#3\@nil#4{%
90   \bb@trim\def\bb@forkv@a{#1}%
91   \bb@trim{\expandafter\bb@kvcmd\expandafter{\bb@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\bb@vforeach#1#2{%
93   \def\bb@forcmd##1{#2}%
94   \bb@fornext#1,\@nil,}
95 \def\bb@fornext#1,{%
96   \ifx\@nil#1\relax\else
97     \bb@ifblank{#1}{\bb@trim\bb@forcmd{#1}}%
98     \expandafter\bb@fornext
99   \fi}
100 \def\bb@foreach#1{\expandafter\bb@vforeach\expandafter{#1}}

```

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

```

101 \def\bb@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}}
103 \def\bb@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 ckecking 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)>> ≡
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)>> ≡

```



```

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#2{%
226   \begingroup
227     \def\{\MessageBreak}%
228     \PackageError{babel}{#1}{#2}%
229   \endgroup}
230 \def\bbbl@warning#1{%
231   \begingroup
232     \def\{\MessageBreak}%
233     \PackageWarning{babel}{#1}%
234   \endgroup}
235 \def\bbbl@infowarn#1{%
236   \begingroup
237     \def\{\MessageBreak}%
238     \PackageNote{babel}{#1}%
239   \endgroup}
240 \def\bbbl@info#1{%
241   \begingroup
242     \def\{\MessageBreak}%
243     \PackageInfo{babel}{#1}%
244   \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.

```

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

```

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

```

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

```

3.3 base

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

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

```

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

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

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

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

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

```
350 \def\bbl@tempa#1=#2\bbl@tempa{%
351   \bbl@csarg\ifx{opt@#1}\@nnil
352     \bbl@csarg\edef{opt@#1}{#2}%
353   \else
354     \bbl@error
355     {Bad option '#1=#2'. Either you have misspelled the\\%
356       key or there is a previous setting of '#1'. Valid\\%
357       keys are, among others, 'shorthands', 'main', 'bidi',\\%
358       'strings', 'config', 'headfoot', 'safe', 'math'.}%
359     {See the manual for further details.}
360   \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.

```
361 \let\bbl@language@opts\@empty
362 \DeclareOption*{%
363   \bbl@xin@{\string=}\CurrentOption}%
364   \ifin@
365     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
366   \else
367     \bbl@add@list\bbl@language@opts{\CurrentOption}%
368   \fi}
```

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

```
369 \ProcessOptions*
370 \ifx\bbl@opt@provide\@nnil
371   \let\bbl@opt@provide\@empty % %%% MOVE above
372 \else
373   \chardef\bbl@iniflag\@ne
374   \bbl@exp{\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
375     \in{,provide,},{, #1,}%
376     \ifin@
377       \def\bbl@opt@provide{#2}%
378       \bbl@replace\bbl@opt@provide{;}{,}%
379     \fi}
380 \fi
381 %
```

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=...`

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

```

388 \fi\fi
389 \expandafter\bb@sh@string
390 \fi}
391 \ifx\bb@opt@shorthands\@nnil
392 \def\bb@ifshorthand#1#2#3{#2}%
393 \else\ifx\bb@opt@shorthands\@empty
394 \def\bb@ifshorthand#1#2#3{#3}%
395 \else

```

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

```

396 \def\bb@ifshorthand#1{%
397 \bb@xin@{\string#1}{\bb@opt@shorthands}%
398 \ifin@
399 \expandafter\@firstoftwo
400 \else
401 \expandafter\@secondoftwo
402 \fi}

```

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

```

403 \edef\bb@opt@shorthands{%
404 \expandafter\bb@sh@string\bb@opt@shorthands\@empty}%

```

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

```

405 \bb@ifshorthand{'}%
406 {\PassOptionsToPackage{activeacute}{babel}}{}
407 \bb@ifshorthand{`}%
408 {\PassOptionsToPackage{activegrave}{babel}}{}
409 \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.

```

410 \ifx\bb@opt@headfoot\@nnil\else
411 \g@addto@macro\@resetactivechars{%
412 \set@typeset@protect
413 \expandafter\select@language@x\expandafter{\bb@opt@headfoot}%
414 \let\protect\noexpand}
415 \fi

```

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

```

416 \ifx\bb@opt@safe\@undefined
417 \def\bb@opt@safe{BR}
418 % \let\bb@opt@safe\@empty % Pending of \cite
419 \fi

```

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

```

420 \bb@trace{Defining IfBabelLayout}
421 \ifx\bb@opt@layout\@nnil
422 \newcommand\IfBabelLayout[3]{#3}%
423 \else
424 \bb@exp{\bb@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
425 \in@{,layout,}{, #1,}%
426 \ifin@
427 \def\bb@opt@layout{#2}%
428 \bb@replace\bb@opt@layout{ }{.}%
429 \fi}
430 \newcommand\IfBabelLayout[1]{%
431 \@expandtwoargs\in@{.#1.}{.\bb@opt@layout.}%
432 \ifin@
433 \expandafter\@firstoftwo
434 \else

```

```

435     \expandafter\@secondoftwo
436     \fi}
437 \fi
438 \</package>
439 \<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*.

```

440 \ifx\ldf@quit\undefined\else
441 \endinput\fi % Same line!
442 <<Make sure ProvidesFile is defined>>
443 \ProvidesFile{babel.def}[\<date>] v\<version> Babel common definitions]
444 \ifx\AtBeginDocument\undefined % TODO. change test.
445   <<Emulate LaTeX>>
446 \fi
447 <<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.

```

448 \</core>
449 \<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.

```

450 \def\bbl@version{\<version>}
451 \def\bbl@date{\<date>}
452 <<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.

```

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

```

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

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

```

468 \def\bbl@fixname#1{%
469   \begingroup
470   \def\bbl@tempe{l@}%

```

```

471 \edef\bbl@tempd{\noexpand\ifundefined{\noexpand\bbl@tempe#1}}%
472 \bbl@tempd
473 {\lowercase\expandafter{\bbl@tempd}%
474 {\uppercase\expandafter{\bbl@tempd}%
475 \@empty
476 {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
477 {\uppercase\expandafter{\bbl@tempd}}}%
478 {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
479 {\lowercase\expandafter{\bbl@tempd}}}%
480 \@empty
481 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
482 \bbl@tempd
483 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}
484 \def\bbl@iflanguage#1{%
485 \ifundefined{#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.

```

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

```

```

528 \def\babelprovide@locale{%
529   \ifx\babelprovide@undefined
530     \babelerror{For a language to be defined on the fly 'base'\\%
531               is not enough, and the whole package must be\\%
532               loaded. Either delete the 'base' option or\\%
533               request the languages explicitly}%
534     {See the manual for further details.}%
535   \fi
536   \let\babel@auxname\language % Still necessary. TODO
537   \babel@ifunset{\babel@bcp@map@\language}{}% Move uplevel??
538   {\edef\language{\@nameuse{\babel@bcp@map@\language}}}%
539   \ifbabel@bcp@allowed
540     \expandafter\ifx\csname date\language\endcsname\relax
541       \expandafter
542       \babel@bcp@lookup\language-\@empty-\@empty-\@empty\@@
543       \ifx\babel@bcp\relax\else % Returned by \babel@bcp@lookup
544         \edef\language{\babel@bcp@prefix\babel@bcp}%
545         \edef\localename{\babel@bcp@prefix\babel@bcp}%
546         \expandafter\ifx\csname date\language\endcsname\relax
547           \let\babel@initoload\babel@bcp
548           \babel@exp{\babelprovide[\babel@autoload@bcptoptions]{\language}}%
549           \let\babel@initoload\relax
550         \fi
551         \babel@csarg\xdef{bcp@map@\babel@bcp}{\localename}%
552       \fi
553     \fi
554   \fi
555   \expandafter\ifx\csname date\language\endcsname\relax
556     \IfFileExists{babel-\language.tex}%
557     {\babel@exp{\babelprovide[\babel@autoload@options]{\language}}}%
558     {}%
559   \fi}
560 (+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.

```

561 \def\iflanguage#1{%
562   \babel@iflanguage{#1}{%
563     \ifnum\csname l@#1\endcsname=\language
564       \expandafter\@firstoftwo
565     \else
566       \expandafter\@secondoftwo
567     \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.

```

568 \let\babel@select@type\z@
569 \edef\selectlanguage{%
570   \noexpand\protect
571   \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`.

```

572 \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.

```

573 \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.

```
574 \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
575 \def\bbl@push@language{%
576   \ifx\language\@undefined\else
577     \ifx\currentgrouplevel\@undefined
578       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
579     \else
580       \ifnum\currentgrouplevel=\z@
581         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
582       \else
583         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
584       \fi
585     \fi
586   \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`.

```
587 \def\bbl@pop@lang#1+#2\@{%
588   \edef\language{#1}%
589   \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).

```
590 \let\bbl@ifrestoring\@secondoftwo
591 \def\bbl@pop@language{%
592   \expandafter\bbl@pop@lang\bbl@language@stack\@
593   \let\bbl@ifrestoring\@firstoftwo
594   \expandafter\bbl@set@language\expandafter{\language}%
595   \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).

```
596 \chardef\localeid\z@
597 \def\bbl@id@last{0} % No real need for a new counter
598 \def\bbl@id@assign{%
599   \bbl@ifunset{bbl@id@\language}%
600   {\count\@bbl@id@last\relax
```

```

601 \advance\count@\@ne
602 \bbl@csarg\chardef{id@@\language}\count@
603 \edef\bbl@id@last{\the\count@}%
604 \ifcase\bbl@engine\or
605 \directlua{
606   Babel = Babel or {}
607   Babel.locale_props = Babel.locale_props or {}
608   Babel.locale_props[\bbl@id@last] = {}
609   Babel.locale_props[\bbl@id@last].name = '\language'
610 }%
611 \fi}%
612 {}%
613 \chardef\localeid\bbl@c{l{id@}}

```

The unprotected part of \selectlanguage.

```

614 \expandafter\def\csname selectlanguage \endcsname#1{%
615 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
616 \bbl@push@language
617 \aftergroup\bbl@pop@language
618 \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).

```

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

```

```

649     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
650     \bbl@savelastskip
651     \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
652     \bbl@restorelastskip
653     \fi
654     \bbl@usehooks{write}}}%
655 \fi
656 \fi}
657 %
658 \let\bbl@restorelastskip\relax
659 \let\bbl@savelastskip\relax
660 %
661 \newif\ifbbl@bcpallowed
662 \bbl@bcpallowedfalse
663 \def\select@language#1{% from set@, babel@aux
664   \ifx\bbl@selectorname\@empty
665     \def\bbl@selectorname{select}%
666     % set hmap
667     \fi
668     \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
669     % set name
670     \edef\languagename{#1}%
671     \bbl@fixname\languagename
672     % TODO. name@map must be here?
673     \bbl@provide@locale
674     \bbl@iflanguage\languagename{%
675       \let\bbl@select@type\z@
676       \expandafter\bbl@switch\expandafter{\languagename}}%
677 \def\babel@aux#1#2{%
678   \select@language{#1}%
679   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
680     \writefile{##1}{\babel@toc{#1}{#2}\relax}}% TODO - plain?
681 \def\babel@toc#1#2{%
682   \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 `\languagename`.

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 `\curname` 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`.

```

683 \newif\ifbbl@usedategroup
684 \let\bbl@savextras\@empty
685 \def\bbl@switch#1{% from select@, foreign@
686   % make sure there is info for the language if so requested
687   \bbl@ensureinfo{#1}%
688   % restore
689   \originalTeX
690   \expandafter\def\expandafter\originalTeX\expandafter{%
691     \curname noextras#1\endcurname
692     \let\originalTeX\@empty
693     \babel@beginsave}%
694   \bbl@usehooks{afterreset}}}%
695   \languageshorthands{none}%
696   % set the locale id

```

```

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

```

```

760 % hyphenation - mins
761 \babel@savevariable\lefthyphenmin
762 \babel@savevariable\righthyphenmin
763 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
764 \set@hyphenmins\tw@\thr@\relax
765 \else
766 \expandafter\expandafter\expandafter\set@hyphenmins
767 \csname #1hyphenmins\endcsname\relax
768 \fi
769 % reset selector name
770 \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.

```

771 \long\def\otherlanguage#1{%
772 \def\bbl@selectorname{other}%
773 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
774 \csname selectlanguage \endcsname{#1}%
775 \ignorespaces}

```

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

```

776 \long\def\endotherlanguage{%
777 \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`.

```

778 \expandafter\def\csname otherlanguage*\endcsname{%
779 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
780 \def\bbl@otherlanguage@s[#1]#2{%
781 \def\bbl@selectorname{other*}%
782 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
783 \def\bbl@select@opts{#1}%
784 \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”.

```

785 \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.

```

786 \providecommand\bbl@beforeforeign{}
787 \edef\foreignlanguage{%
788   \noexpand\protect
789   \expandafter\noexpand\csname foreignlanguage \endcsname}
790 \expandafter\def\csname foreignlanguage \endcsname{%
791   \@ifstar\bbl@foreign@s\bbl@foreign@x}
792 \providecommand\bbl@foreign@x[3][[]]{%
793   \begingroup
794     \def\bbl@selectorname{foreign}%
795     \def\bbl@select@opts{#1}%
796     \let\BabelText\@firstofone
797     \bbl@beforeforeign
798     \foreign@language{#2}%
799     \bbl@usehooks{foreign}{}%
800     \BabelText{#3}% Now in horizontal mode!
801   \endgroup}
802 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
803   \begingroup
804     {\par}%
805     \def\bbl@selectorname{foreign*}%
806     \let\bbl@select@opts\@empty
807     \let\BabelText\@firstofone
808     \foreign@language{#1}%
809     \bbl@usehooks{foreign*}{}%
810     \bbl@dirparastext
811     \BabelText{#2}% Still in vertical mode!
812   {\par}%
813   \endgroup}

```

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

```

814 \def\foreign@language#1{%
815   % set name
816   \edef\language#1%
817   \ifbbl@usedategroup
818     \bbl@add\bbl@select@opts{,date,}%
819     \bbl@usedategroupfalse
820   \fi
821   \bbl@fixname\language
822   % TODO. name@map here?
823   \bbl@provide@locale
824   \bbl@iflanguage\language#1%
825   \let\bbl@select@type\@ne
826   \expandafter\bbl@switch\expandafter{\language}

```

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

```

827 \def\IfBabelSelectorTF#1{%
828   \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
829   \ifin@
830     \expandafter\@firstoftwo
831   \else
832     \expandafter\@secondoftwo
833   \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.

```

834 \let\bbl@hyphlist\@empty
835 \let\bbl@hyphenation@\relax
836 \let\bbl@pttnlist\@empty
837 \let\bbl@patterns@relax
838 \let\bbl@hymapset=\@cclv
839 \def\bbl@patterns#1{%
840   \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
841     \csname l@#1\endcsname
842     \edef\bbl@tempa{#1}%
843   \else
844     \csname l@#1:f@encoding\endcsname
845     \edef\bbl@tempa{#1:f@encoding}%
846   \fi
847   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
848   % > luatex
849   \@ifundefined{bbl@hyphenation@}{% Can be \relax!
850     \begingroup
851       \bbl@xin@{, \number\language,}{, \bbl@hyphlist}%
852     \ifin@else
853       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
854       \hyphenation{%
855         \bbl@hyphenation@
856         \@ifundefined{bbl@hyphenation@#1}%
857         \@empty
858         {\space\csname bbl@hyphenation@#1\endcsname}}%
859       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
860     \fi
861   \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*`.

```

862 \def\hyphenrules#1{%
863   \edef\bbl@tempf{#1}%
864   \bbl@fixname\bbl@tempf
865   \bbl@iflanguage\bbl@tempf{%
866     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
867     \ifx\languageshorthands\@undefined\else
868       \languageshorthands{none}%
869     \fi
870     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
871       \set@hyphenmins\tw@\thr@@\relax
872     \else
873       \expandafter\expandafter\expandafter\set@hyphenmins
874       \csname\bbl@tempf hyphenmins\endcsname\relax
875     \fi}}
876 \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.

```

877 \def\providehyphenmins#1#2{%
878   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
879     \@namedef{#1hyphenmins}{#2}%
880   \fi}
```

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

```

881 \def\set@hyphenmins#1#2{%
```

```

882 \lefthyphenmin#1\relax
883 \righthyphenmin#2\relax}

```

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

```

884 \ifx\ProvidesFile\undefined
885   \def\ProvidesLanguage#1[#2 #3 #4]{%
886     \wlog{Language: #1 #4 #3 <#2>}%
887   }
888 \else
889   \def\ProvidesLanguage#1{%
890     \begingroup
891       \catcode`\ 10 %
892       \@makeother\/%
893       \@ifnextchar[%]
894         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
895   \def\@provideslanguage#1[#2]{%
896     \wlog{Language: #1 #2}%
897     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
898     \endgroup}
899 \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`.

```

900 \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.

```

901 \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’:

```

902 \providecommand\setlocale{%
903   \bbl@error
904     {Not yet available}%
905     {Find an armchair, sit down and wait}}
906 \let\uselocale\setlocale
907 \let\locale\setlocale
908 \let\selectlocale\setlocale
909 \let\textlocale\setlocale
910 \let\textlanguage\setlocale
911 \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_ϵ , so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

```

912 \edef\bbl@nulllanguage{\string\language=0}
913 \def\bbl@nocaption{\protect\bbl@nocaption@i}
914 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
915   \global\@namedef{#2}{\textbf{?#1?}}}%
916   \@nameuse{#2}%

```



```

917 \edef\bbl@tempa{#1}%
918 \bbl@sreplace\bbl@tempa{name}{}%
919 \bbl@warning{%
920   \@backslashchar#1 not set for '\language'. Please,\\%
921   define it after the language has been loaded\\%
922   (typically in the preamble) with:\\%
923   \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
924   Feel free to contribute on github.com/latex3/babel.\\%
925   Reported}}
926 \def\bbl@tentative{\protect\bbl@tentative@i}
927 \def\bbl@tentative@i#1{%
928   \bbl@warning{%
929     Some functions for '#1' are tentative.\\%
930     They might not work as expected and their behavior\\%
931     could change in the future.\\%
932     Reported}}
933 \def\@nolanerr#1{%
934   \bbl@error
935   {You haven't defined the language '#1' yet.\\%
936     Perhaps you misspelled it or your installation\\%
937     is not complete}%
938   {Your command will be ignored, type <return> to proceed}}
939 \def\@nopatterns#1{%
940   \bbl@warning
941   {No hyphenation patterns were preloaded for\\%
942     the language '#1' into the format.\\%
943     Please, configure your TeX system to add them and\\%
944     rebuild the format. Now I will use the patterns\\%
945     preloaded for \bbl@nulllanguage\space instead}}
946 \let\bbl@usehooks\@gobbletwo
947 \ifx\bbl@onlyswitch\@empty\endinput\fi
948 % Here ended switch.def

```

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

```

949 \ifx\directlua\@undefined\else
950   \ifx\bbl@luapatterns\@undefined
951     \input luababel.def
952   \fi
953 \fi
954 \bbl@trace{Compatibility with language.def}
955 \ifx\bbl@languages\@undefined
956   \ifx\directlua\@undefined
957     \openin1 = language.def % TODO. Remove hardcoded number
958     \ifeof1
959       \closein1
960       \message{I couldn't find the file language.def}
961     \else
962       \closein1
963       \begingroup
964         \def\addlanguage#1#2#3#4#5{%
965           \expandafter\ifx\csname lang@#1\endcsname\relax\else
966             \global\expandafter\let\csname l@#1\expandafter\endcsname
967             \csname lang@#1\endcsname
968           \fi}%
969         \def\uselanguage#1{%
970           \input language.def
971         \endgroup
972       \fi
973     \fi
974     \chardef\l@english\z@
975 \fi

```

\addto It takes two arguments, a *<control sequence>* and T_EX-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.

```

976 \def\addto#1#2{%
977   \ifx#1\undefined
978     \def#1{#2}%
979   \else
980     \ifx#1\relax
981       \def#1{#2}%
982     \else
983       {\toks@\expandafter{#1#2}%
984        \xdef#1{\the\toks@}}%
985     \fi
986   \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.

```

987 \def\bbl@withactive#1#2{%
988   \beginingroup
989     \lccode`~=#2\relax
990     \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`.

```

991 \def\bbl@redefine#1{%
992   \edef\bbl@tempa{\bbl@stripslash#1}%
993   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
994   \expandafter\def\csname\bbl@tempa\endcsname}
995 \@onlypreamble\bbl@redefine

```

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

```

996 \def\bbl@redefine@long#1{%
997   \edef\bbl@tempa{\bbl@stripslash#1}%
998   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
999   \long\expandafter\def\csname\bbl@tempa\endcsname}
1000 \@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_`.

```

1001 \def\bbl@redefineroobust#1{%
1002   \edef\bbl@tempa{\bbl@stripslash#1}%
1003   \bbl@ifunset{\bbl@tempa\space}%
1004     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1005      \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1006     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1007   \@namedef{\bbl@tempa\space}{}
1008 \@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.

```

1009 \bbl@trace{Hooks}
1010 \newcommand\AddBabelHook[3][[]]{%
1011   \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}{}}%

```

```

1012 \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1013 \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1014 \bbl@ifunset{\bbl@ev@#2@#3@#1}%
1015 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1016 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1017 \bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
1018 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1019 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1020 \def\bbl@usehooks{\bbl@usehooks@lang\language}
1021 \def\bbl@usehooks@lang#1#2#3{% Test for Plain
1022 \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1023 \def\bbl@elth##1{%
1024 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}}%
1025 \bbl@cs{ev@#2@#3}%
1026 \ifx\language\@undefined\else % Test required for Plain (?)
1027 \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1028 \def\bbl@elth##1{%
1029 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#3}}}%
1030 \bbl@cs{ev@#2@#3}%
1031 \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).

```

1032 \def\bbl@evargs{,% <- don't delete this comma
1033 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1034 adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1035 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1036 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1037 beforestart=0,language=2,begindocument=1}
1038 \ifx\NewHook\@undefined\else % Test for Plain (?)
1039 \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1040 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1041 \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.

```

1042 \bbl@trace{Defining babelensure}
1043 \newcommand\babelensure[2][{}]{%
1044 \AddBabelHook{babel-ensure}{afterextras}{%
1045 \ifcase\bbl@select@type
1046 \bbl@cl{e}%
1047 \fi}%
1048 \begingroup
1049 \let\bbl@ens@include\@empty
1050 \let\bbl@ens@exclude\@empty
1051 \def\bbl@ens@fontenc{\relax}%
1052 \def\bbl@tempb##1{%
1053 \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1054 \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1055 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ens@##1}{##2}}%
1056 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1057 \def\bbl@tempc{\bbl@ensure}%
1058 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1059 \expandafter{\bbl@ens@include}}%
1060 \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%

```

```

1061 \expandafter{\bbl@ens@exclude}}%
1062 \toks@\expandafter{\bbl@tempc}%
1063 \bbl@exp{%
1064 \endgroup
1065 \def<bbl@e#2>{\the\toks@{\bbl@ens@fontenc}}%
1066 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1067 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1068 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1069 \edef##1{\noexpand\bbl@nocaption
1070 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}%
1071 \fi
1072 \ifx##1\empty\else
1073 \in@{##1}{#2}%
1074 \ifin\else
1075 \bbl@ifunset{\bbl@ensure@\language\language}%
1076 {\bbl@exp{%
1077 \\\DeclareRobustCommand<bbl@ensure@\language\language>[1]{%
1078 \\\foreignlanguage{\language\language}%
1079 {\ifx\relax#3\else
1080 \\\fontencoding{#3}\selectfont
1081 \fi
1082 #####1}}}%
1083 }%
1084 \toks@\expandafter{##1}%
1085 \edef##1{%
1086 \bbl@csarg\noexpand{ensure@\language\language}%
1087 {\the\toks@}}%
1088 \fi
1089 \expandafter\bbl@tempb
1090 \fi}%
1091 \expandafter\bbl@tempb\bbl@captionslist\today\empty
1092 \def\bbl@tempa##1{% elt for include list
1093 \ifx##1\empty\else
1094 \bbl@csarg\in@{ensure@\language\language\expandafter}\expandafter{##1}%
1095 \ifin\else
1096 \bbl@tempb##1\empty
1097 \fi
1098 \expandafter\bbl@tempa
1099 \fi}%
1100 \bbl@tempa#1\empty}
1101 \def\bbl@captionslist{%
1102 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1103 \contentsname\listfigurename\listtablename\indexname\figurename
1104 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1105 \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.

```

1106 \bbl@trace{Macros for setting language files up}
1107 \def\bbl@ldfinit{%
1108   \let\bbl@screset\@empty
1109   \let\BabelStrings\bbl@opt@string
1110   \let\BabelOptions\@empty
1111   \let\BabelLanguages\relax
1112   \ifx\originalTeX\@undefined
1113     \let\originalTeX\@empty
1114   \else
1115     \originalTeX
1116   \fi}
1117 \def\LdfInit#1#2{%
1118   \chardef\atcatcode=\catcode`\@
1119   \catcode`\@=11\relax
1120   \chardef\eqcatcode=\catcode`\=
1121   \catcode`\==12\relax
1122   \expandafter\if\expandafter\@backslashchar
1123     \expandafter\@car\string#2\@nil
1124     \ifx#2\@undefined\else
1125       \ldf@quit{#1}%
1126     \fi
1127   \else
1128     \expandafter\ifx\csname#2\endcsname\relax\else
1129       \ldf@quit{#1}%
1130     \fi
1131   \fi
1132   \bbl@ldfinit}

```

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

```

1133 \def\ldf@quit#1{%
1134   \expandafter\main@language\expandafter{#1}%
1135   \catcode`\@=\atcatcode \let\atcatcode\relax
1136   \catcode`\==\eqcatcode \let\eqcatcode\relax
1137   \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.

```

1138 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1139   \bbl@afterlang
1140   \let\bbl@afterlang\relax
1141   \let\BabelModifiers\relax
1142   \let\bbl@screset\relax}%
1143 \def\ldf@finish#1{%
1144   \loadlocalcfg{#1}%
1145   \bbl@afterldf{#1}%
1146   \expandafter\main@language\expandafter{#1}%
1147   \catcode`\@=\atcatcode \let\atcatcode\relax
1148   \catcode`\==\eqcatcode \let\eqcatcode\relax}

```

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

```

1149 \@onlypreamble\LdfInit
1150 \@onlypreamble\ldf@quit
1151 \@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.

```

1152 \def\main@language#1{%
1153   \def\bbl@main@language{#1}%
1154   \let\language\main@language % TODO. Set localename
1155   \bbl@id@assign
1156   \bbl@patterns{\language}%

```

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

```

1157 \def\bbl@beforestart{%
1158   \def\@nolanerr##1{%
1159     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1160   \bbl@usehooks{beforestart}{}%
1161   \global\let\bbl@beforestart\relax}
1162 \AtBeginDocument{%
1163   {\@nameuse{bbl@beforestart}}% Group!
1164   \if@filesw
1165     \providecommand\babel@aux[2]{}%
1166     \immediate\write\@mainaux{%
1167       \string\providecommand\string\babel@aux[2]{}%
1168       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1169     \fi
1170 \<-package>
1171 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1172 \<+package>
1173 \ifbbl@single % must go after the line above.
1174   \renewcommand\selectlanguage[1]{}%
1175   \renewcommand\foreignlanguage[2]{#2}%
1176   \global\let\babel@aux\@gobbletwo % Also as flag
1177 \fi}
1178 \<-core>
1179 \AddToHook{begindocument/before}{%
1180   \expandafter\selectlanguage\expandafter{\bbl@main@language}}
1181 \<+core>
1182 \ifcase\bbl@engine\or
1183   \AtBeginDocument{\pagedir\bodydir} % TODO - a better place
1184 \fi

```

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

```

1185 \def\select@language@x#1{%
1186   \ifcase\bbl@select@type
1187     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1188   \else
1189     \select@language{#1}%
1190   \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.

```

1191 \bbl@trace{Shorhands}
1192 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1193   \bbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1194   \bbl@ifunset{\@sanitize}{\bbl@add\@sanitize{\@makeother#1}}%
1195   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1196     \begingroup

```

```

1197 \catcode`#1\active
1198 \nfss@catcodes
1199 \ifnum\catcode`#1=\active
1200 \endgroup
1201 \bbl@add\nfss@catcodes{\@makeother#1}%
1202 \else
1203 \endgroup
1204 \fi
1205 \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.

```

1206 \def\bbl@remove@special#1{%
1207 \begingroup
1208 \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1209 \else\noexpand##1\noexpand##2\fi}%
1210 \def\do{\x\do}%
1211 \def\@makeother{\x\@makeother}%
1212 \edef\x{\endgroup
1213 \def\noexpand\dospecials{\dospecials}%
1214 \expandafter\ifx\csname @sanitize\endcsname\relax\else
1215 \def\noexpand\@sanitize{\@sanitize}%
1216 \fi}%
1217 \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).

```

1218 \def\bbl@active@def#1#2#3#4{%
1219 \@namedef{#3#1}{%
1220 \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1221 \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1222 \else
1223 \bbl@afterfi\csname#2@sh@#1\endcsname
1224 \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.

```

1225 \long\@namedef{#3@arg#1}##1{%
1226 \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1227 \bbl@afterelse\csname#4#1\endcsname##1%
1228 \else
1229 \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1230 \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.

```

1231 \def\initiate@active@char#1{%
1232   \bbl@ifunset{active@char\string#1}%
1233   {\bbl@withactive
1234    {\expandafter\@initiate@active@char\expandafter}\#1\string#1}%
1235   {}}

```

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).

```

1236 \def\@initiate@active@char#1#2#3{%
1237   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1238   \ifx#1\@undefined
1239     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}}%
1240   \else
1241     \bbl@csarg\let{oridef@#2}#1%
1242     \bbl@csarg\edef{oridef@#2}{%
1243       \let\noexpand#1%
1244       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1245   \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`).

```

1246   \ifx#1#3\relax
1247     \expandafter\let\csname normal@char#2\endcsname#3%
1248   \else
1249     \bbl@info{Making #2 an active character}%
1250     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1251     \@namedef{normal@char#2}{%
1252       \textormath{#3}{\csname bbl@oridef@#2\endcsname}}}%
1253   \else
1254     \@namedef{normal@char#2}{#3}%
1255   \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).

```

1256   \bbl@restoreactive{#2}%
1257   \AtBeginDocument{%
1258     \catcode`#2\active
1259     \if@filesw
1260       \immediate\write\@mainaux{\catcode`\string#2\active}%
1261     \fi}%
1262   \expandafter\bbl@add@special\csname#2\endcsname
1263   \catcode`#2\active
1264   \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⟩`).

```

1265   \let\bbl@tempa\@firstoftwo
1266   \if\string^#2%
1267     \def\bbl@tempa{\noexpand\textormath}%
1268   \else
1269     \ifx\bbl@mathnormal\@undefined\else
1270       \let\bbl@tempa\bbl@mathnormal
1271     \fi

```



```

1272 \fi
1273 \expandafter\edef\csname active@char#2\endcsname{%
1274   \bbl@tempa
1275   {\noexpand\if@safe@actives
1276     \noexpand\expandafter
1277     \expandafter\noexpand\csname normal@char#2\endcsname
1278     \noexpand\else
1279     \noexpand\expandafter
1280     \expandafter\noexpand\csname bbl@doactive#2\endcsname
1281     \noexpand\fi}%
1282   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1283 \bbl@csarg\edef{doactive#2}{%
1284   \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

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

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

```

1285 \bbl@csarg\edef{active@#2}{%
1286   \noexpand\active@prefix\noexpand#1%
1287   \expandafter\noexpand\csname active@char#2\endcsname}%
1288 \bbl@csarg\edef{normal@#2}{%
1289   \noexpand\active@prefix\noexpand#1%
1290   \expandafter\noexpand\csname normal@char#2\endcsname}%
1291 \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.

```

1292 \bbl@active@def#2\user@group{user@active}{language@active}%
1293 \bbl@active@def#2\language@group{language@active}{system@active}%
1294 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1295 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1296   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1297 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1298   {\expandafter\noexpand\csname user@active#2\endcsname}%

```

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

```

1299 \if\string'#2%
1300   \let\prim@s\bbl@prim@s
1301   \let\active@math@prime#1%
1302 \fi
1303 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1304 <<(*More package options)>> \equiv
1305 \DeclareOption{math=active}{}
1306 \DeclareOption{math=normal}{{\def\bbl@mathnormal{\noexpand\textormath}}}
1307 <</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*.

```

1308 \@ifpackagewith{babel}{KeepShorthandsActive}%
1309 {\let\bbl@restoreactive\@gobble}%
1310 {\def\bbl@restoreactive#1{%
1311     \bbl@exp{%
1312         \\AfterBabelLanguage\\CurrentOption
1313         {\catcode`#1=\the\catcode`#1\relax}%
1314         \\AtEndOfPackage
1315         {\catcode`#1=\the\catcode`#1\relax}}}%
1316     \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.

```

1317 \def\bbl@sh@select#1#2{%
1318     \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1319         \bbl@afterelse\bbl@scndcs
1320     \else
1321         \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1322     \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 `\ifincsn` is available. If there is, the expansion will be more robust.

```

1323 \begingroup
1324 \bbl@ifunset{ifincsn}% TODO. Ugly. Correct? Only Plain?
1325 {\gdef\active@prefix#1{%
1326     \ifx\protect\@typeset@protect
1327     \else
1328         \ifx\protect\@unexpandable@protect
1329             \noexpand#1%
1330         \else
1331             \protect#1%
1332         \fi
1333         \expandafter\@gobble
1334     \fi}}
1335 {\gdef\active@prefix#1{%
1336     \ifincsn
1337         \string#1%
1338         \expandafter\@gobble
1339     \else
1340         \ifx\protect\@typeset@protect
1341         \else
1342             \ifx\protect\@unexpandable@protect
1343                 \noexpand#1%
1344             \else
1345                 \protect#1%
1346             \fi
1347             \expandafter\expandafter\expandafter\@gobble
1348         \fi
1349     \fi}}
1350 \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@activestrue`), 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@activesfalse`).

```
1351 \newif\if@safe@actives
1352 \@safe@activesfalse
```

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

```
1353 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\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`.

```
1354 \chardef\bbl@activated\z@
1355 \def\bbl@activate#1{%
1356   \chardef\bbl@activated\@ne
1357   \bbl@withactive{\expandafter\let\expandafter}#1%
1358   \csname bbl@active@\string#1\endcsname}
1359 \def\bbl@deactivate#1{%
1360   \chardef\bbl@activated\tw@
1361   \bbl@withactive{\expandafter\let\expandafter}#1%
1362   \csname bbl@normal@\string#1\endcsname}
```

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

```
\bbl@scndcs
1363 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1364 \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.

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

```

1388 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb@firstcs
1389 \bb@ifunset{#1@sh@\string#2@\string#3@}{}%
1390 {\def\bb@tempa{#4}%
1391 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bb@tempa
1392 \else
1393 \bb@info
1394 {Redefining #1 shorthand \string#2\string#3\\%
1395 in language \CurrentOption}%
1396 \fi}%
1397 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1398 \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.

```

1399 \def\textormath{%
1400 \ifmmode
1401 \expandafter\@secondoftwo
1402 \else
1403 \expandafter\@firstoftwo
1404 \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` name of the level or group is stored in a macro. The default is to have a user group; use language `\system@group` group ‘english’ and have a system group called ‘system’.

```

1405 \def\user@group{user}
1406 \def\language@group{english} % TODO. I don't like defaults
1407 \def\system@group{system}

```

`\usesshorthands` 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.

```

1408 \def\usesshorthands{%
1409 \ifstar\bb@usesesh@s{\bb@usesesh@x{}}%
1410 \def\bb@usesesh@s#1{%
1411 \bb@usesesh@x
1412 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb@activate{#1}}}%
1413 {#1}}
1414 \def\bb@usesesh@x#1#2{%
1415 \bb@ifshorthand{#2}%
1416 {\def\user@group{user}%
1417 \initiate@active@char{#2}%
1418 #1%
1419 \bb@activate{#2}}%
1420 {\bb@error
1421 {I can't declare a shorthand turned off (\string#2)}
1422 {Sorry, but you can't use shorthands which have been\\%
1423 turned off in the package options}}}

```

`\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.

```

1424 \def\user@language@group{user@\language@group}
1425 \def\bb@set@user@generic#1#2{%
1426 \bb@ifunset{user@generic@active#1}%
1427 {\bb@active@def#1\user@language@group{user@active}{user@generic@active}%
1428 \bb@active@def#1\user@group{user@generic@active}{language@active}%
1429 \expandafter\edef\csname#2@sh@#1@\endcsname{%
1430 \expandafter\noexpand\csname normal@char#1\endcsname}%
1431 \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1432 \expandafter\noexpand\csname user@active#1\endcsname}}%

```

```

1433 \@empty}
1434 \newcommand\defineshorthand[3][user]{%
1435 \edef\bbl@tempa{\zap@space#1 \@empty}%
1436 \bbl@for\bbl@tempb\bbl@tempa{%
1437 \if*\expandafter\@car\bbl@tempb\@nil
1438 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1439 \@expandtwoargs
1440 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1441 \fi
1442 \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

`\languageshorthands` A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

```

1443 \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 latest to `\active@char`.

```

1444 \def\aliasshorthand#1#2{%
1445 \bbl@ifshorthand{#2}%
1446 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1447 \ifx\document\@notprerr
1448 \@notshorthand{#2}%
1449 \else
1450 \initiate@active@char{#2}%
1451 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1452 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1453 \bbl@activate{#2}%
1454 \fi
1455 \fi}%
1456 {\bbl@error
1457 {Cannot declare a shorthand turned off (\string#2)}
1458 {Sorry, but you cannot use shorthands which have been\\%
1459 turned off in the package options}}}

```

`\@notshorthand`

```

1460 \def\@notshorthand#1{%
1461 \bbl@error{%
1462 The character '\string #1' should be made a shorthand character;\\%
1463 add the command \string\usesshorthands\string{#1\string} to
1464 the preamble.\\%
1465 I will ignore your instruction}%
1466 {You may proceed, but expect unexpected results}}

```

`\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.

```

1467 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1468 \DeclareRobustCommand*\shorthandoff{%
1469 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1470 \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.

```

1471 \def\bbl@switch@sh#1#2{%
1472 \ifx#2\@nnil\else
1473 \bbl@ifunset{\bbl@active@\string#2}%

```

```

1474 {\bbl@error
1475 {I can't switch '\string#2' on or off--not a shorthand}%
1476 {This character is not a shorthand. Maybe you made\\%
1477 a typing mistake? I will ignore your instruction.}}%
1478 {\ifcase#1% off, on, off*
1479 \catcode`#212\relax
1480 \or
1481 \catcode`#2\active
1482 \bbl@ifunset{bbl@shdef@\string#2}%
1483 {}%
1484 {\bbl@withactive{\expandafter\let\expandafter}#2%
1485 \csname bbl@shdef@\string#2\endcsname
1486 \bbl@csarg\let{shdef@\string#2}\relax}%
1487 \ifcase\bbl@activated\or
1488 \bbl@activate{#2}%
1489 \else
1490 \bbl@deactivate{#2}%
1491 \fi
1492 \or
1493 \bbl@ifunset{bbl@shdef@\string#2}%
1494 {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1495 {}%
1496 \csname bbl@oricat@\string#2\endcsname
1497 \csname bbl@oridef@\string#2\endcsname
1498 \fi}%
1499 \bbl@afterfi\bbl@switch@sh#1%
1500 \fi}

```

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

```

1501 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1502 \def\bbl@putsh#1{%
1503 \bbl@ifunset{bbl@active@\string#1}%
1504 {\bbl@putsh@i#1@empty\@nnil}%
1505 {\csname bbl@active@\string#1\endcsname}}
1506 \def\bbl@putsh@i#1#2\@nnil{%
1507 \csname\language@group @sh@\string#1@%
1508 \ifx\@empty#2\else\string#2@fi\endcsname}
1509 %
1510 \ifx\bbl@opt@shorthands\@nnil\else
1511 \let\bbl@s@initiate@active@char\initiate@active@char
1512 \def\initiate@active@char#1{%
1513 \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1514 \let\bbl@s@switch@sh\bbl@switch@sh
1515 \def\bbl@switch@sh#1#2{%
1516 \ifx#2\@nnil\else
1517 \bbl@afterfi
1518 \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1519 \fi}
1520 \let\bbl@s@activate\bbl@activate
1521 \def\bbl@activate#1{%
1522 \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1523 \let\bbl@s@deactivate\bbl@deactivate
1524 \def\bbl@deactivate#1{%
1525 \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1526 \fi

```

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

```

1527 \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.

```

1528 \def\bbl@prim@s{%
1529   \prime\futurelet\@let@token\bbl@pr@m@s}
1530 \def\bbl@if@primes#1#2{%
1531   \ifx#1\@let@token
1532     \expandafter\@firstoftwo
1533   \else\ifx#2\@let@token
1534     \bbl@afterelse\expandafter\@firstoftwo
1535   \else
1536     \bbl@afterfi\expandafter\@secondoftwo
1537   \fi\fi}
1538 \begingroup
1539   \catcode`\^=7 \catcode`\*= \active \lccode`\*=`^
1540   \catcode`\'=12 \catcode`\`= \active \lccode`\`= ` '
1541   \lowercase{%
1542     \gdef\bbl@pr@m@s{%
1543       \bbl@if@primes" '%
1544         \pr@@s
1545         {\bbl@if@primes*^ \pr@@t\egroup}}
1546 \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).

```

1547 \initiate@active@char{~}
1548 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1549 \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.

```

1550 \expandafter\def\csname OT1dqpos\endcsname{127}
1551 \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

```

1552 \ifx\f@encoding\@undefined
1553   \def\f@encoding{OT1}
1554 \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.

```

1555 \bbl@trace{Language attributes}
1556 \newcommand\languageattribute[2]{%
1557   \def\bbl@tempc{#1}%
1558   \bbl@fixname\bbl@tempc
1559   \bbl@iflanguage\bbl@tempc{%
1560     \bbl@vforeach{#2}{%

```

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

```

1561     \ifx\bbl@known@attribs\@undefined
1562       \in@false
1563     \else
1564       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%

```

```

1565 \fi
1566 \ifin@
1567 \bbl@warning{%
1568     You have more than once selected the attribute '##1'\%
1569     for language #1. Reported}%
1570 \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 T_EX-code.

```

1571 \bbl@exp{%
1572     \\\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1573 \edef\bbl@tempa{\bbl@tempc-##1}%
1574 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1575 {\csname\bbl@tempc @attr##1\endcsname}%
1576 {\@attrerr{\bbl@tempc}{##1}}%
1577 \fi}}
1578 \@onlypreamble\languageattribute

```

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

```

1579 \newcommand*{\@attrerr}[2]{%
1580 \bbl@error
1581 {The attribute #2 is unknown for language #1.}%
1582 {Your command will be ignored, type <return> to proceed}}

```

\bbl@declare@ttribute This command adds the new language/attribute combination to the list of known attributes. Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro `\extras...` for the current language is extended, otherwise the attribute will not work as its code is removed from memory at `\begin{document}`.

```

1583 \def\bbl@declare@ttribute#1#2#3{%
1584 \bbl@xin@{,#2,}{,\BabelModifiers,}%
1585 \ifin@
1586 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1587 \fi
1588 \bbl@add@list\bbl@attributes{#1-#2}%
1589 \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret T_EX 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.

```

1590 \def\bbl@ifattributeset#1#2#3#4{%
1591 \ifx\bbl@known@attribs\@undefined
1592 \in@false
1593 \else
1594 \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1595 \fi
1596 \ifin@
1597 \bbl@afterelse#3%
1598 \else
1599 \bbl@afterfi#4%
1600 \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 T_EX-code to be executed when the attribute is known and the T_EX-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.

```

1601 \def\bbl@ifknown@ttrib#1#2{%
1602 \let\bbl@tempa\@secondoftwo
1603 \bbl@loopx\bbl@tempb{#2}{%
1604 \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1605 \ifin@

```



```

1606 \let\bbl@tempa\@firstoftwo
1607 \else
1608 \fi}%
1609 \bbl@tempa}

```

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

```

1610 \def\bbl@clear@ttribs{%
1611 \ifx\bbl@attributes\undefined\else
1612 \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1613 \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1614 \let\bbl@attributes\undefined
1615 \fi}
1616 \def\bbl@clear@ttrib#1-#2.{%
1617 \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1618 \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`

```

1619 \bbl@trace{Macros for saving definitions}
1620 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1621 \newcount\babel@savecnt
1622 \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.

```

1623 \def\babel@save#1{%
1624 \def\bbl@tempa{{, #1,}}% Clumsy, for Plain
1625 \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1626 \expandafter{\expandafter,\bbl@savedextras,}}%
1627 \expandafter\in@\bbl@tempa
1628 \ifin@else
1629 \bbl@add\bbl@savedextras{, #1,}%
1630 \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1631 \toks@\expandafter{\originalTeX\let#1=}
1632 \bbl@exp{%
1633 \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1634 \advance\babel@savecnt\@ne
1635 \fi}
1636 \def\babel@savevariable#1{%
1637 \toks@\expandafter{\originalTeX #1=}
1638 \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 on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an

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

auxiliary macro is defined, but the main part is in `\babelprovide`. This new method should be ideally the default one.

```

1639 \def\bbbl@frenchspacing{%
1640   \ifnum\the\sfcode`\.\=@m
1641     \let\bbbl@nonfrenchspacing\relax
1642   \else
1643     \frenchspacing
1644     \let\bbbl@nonfrenchspacing\nonfrenchspacing
1645   \fi}
1646 \let\bbbl@nonfrenchspacing\nonfrenchspacing
1647 \let\bbbl@elt\relax
1648 \edef\bbbl@fs@chars{%
1649   \bbbl@elt{\string.}\@m{3000}\bbbl@elt{\string?}\@m{3000}%
1650   \bbbl@elt{\string!}\@m{3000}\bbbl@elt{\string:}\@m{2000}%
1651   \bbbl@elt{\string;}\@m{1500}\bbbl@elt{\string,}\@m{1250}}
1652 \def\bbbl@pre@fs{%
1653   \def\bbbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1654   \edef\bbbl@save@sfcodes{\bbbl@fs@chars}%
1655 \def\bbbl@post@fs{%
1656   \bbbl@save@sfcodes
1657   \edef\bbbl@tempa{\bbbl@cl{frspc}}%
1658   \edef\bbbl@tempa{\expandafter\@car\bbbl@tempa\@nil}%
1659   \if u\bbbl@tempa      % do nothing
1660   \else\if n\bbbl@tempa  % non french
1661     \def\bbbl@elt##1##2##3{%
1662       \ifnum\sfcode`##1=##2\relax
1663         \babel@savevariable{\sfcode`##1}%
1664         \sfcode`##1=##3\relax
1665       \fi}%
1666     \bbbl@fs@chars
1667   \else\if y\bbbl@tempa  % french
1668     \def\bbbl@elt##1##2##3{%
1669       \ifnum\sfcode`##1=##3\relax
1670         \babel@savevariable{\sfcode`##1}%
1671         \sfcode`##1=##2\relax
1672       \fi}%
1673     \bbbl@fs@chars
1674   \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.

```

1675 \bbbl@trace{Short tags}
1676 \def\babeltags#1{%
1677   \edef\bbbl@tempa{\zap@space#1 \@empty}%
1678   \def\bbbl@tempb##1=##2\@{#}%
1679   \edef\bbbl@tempc{%
1680     \noexpand\newcommand
1681     \expandafter\noexpand\csname ##1\endcsname{%
1682       \noexpand\protect
1683       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1684     \noexpand\newcommand
1685     \expandafter\noexpand\csname text##1\endcsname{%
1686       \noexpand\foreignlanguage{##2}}
1687   \bbbl@tempc}%
1688   \bbbl@for\bbbl@tempa\bbbl@tempa{%
1689     \expandafter\bbbl@tempb\bbbl@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.

```

1690 \bbl@trace{Hyphens}
1691 \@onlypreamble\babelhyphenation
1692 \AtEndOfPackage{%
1693   \newcommand\babelhyphenation[2][\@empty]{%
1694     \ifx\bbl@hyphenation@relax
1695       \let\bbl@hyphenation@ \@empty
1696     \fi
1697     \ifx\bbl@hyphlist \@empty \else
1698       \bbl@warning{%
1699         You must not intermingle \string\selectlanguage\space and\\%
1700         \string\babelhyphenation\space or some exceptions will not\\%
1701         be taken into account. Reported}%
1702       \fi
1703     \ifx \@empty #1%
1704       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1705     \else
1706       \bbl@vforeach{#1}{%
1707         \def\bbl@tempa{##1}%
1708         \bbl@fixname\bbl@tempa
1709         \bbl@iflanguage\bbl@tempa{%
1710           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1711             \bbl@ifunset\bbl@hyphenation@\bbl@tempa}%
1712             {}%
1713             {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1714             #2}}}%
1715       \fi}}

```

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

```

1716 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1717 \def\bbl@t@one{T1}
1718 \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`.

```

1719 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1720 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1721 \def\bbl@hyphen{%
1722   \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1723 \def\bbl@hyphen@i#1#2{%
1724   \bbl@ifunset\bbl@hy@#1#2\@empty}%
1725   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1726   {\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.

```

1727 \def\bbl@usehyphen#1{%
1728   \leavevmode
1729   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1730   \nobreak\hskip\z@skip}

```

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

```

1731 \def\bbl@usehyphen#1{%
1732   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

1733 \def\bbl@hyphenchar{%
1734   \ifnum\hyphenchar\font=\m@ne
1735     \babe\nullhyphen
1736   \else
1737     \char\hyphenchar\font
1738   \fi}

```

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

```

1739 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1740 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1741 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1742 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1743 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1744 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1745 \def\bbl@hy@repeat{%
1746   \bbl@usehyphen%
1747   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1748 \def\bbl@hy@repeat{%
1749   \bbl@usehyphen%
1750   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1751 \def\bbl@hy@empty{\hskip\z@skip}
1752 \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.

```

1753 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

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.

```

1754 \bbl@trace{Multiencoding strings}
1755 \def\bbl@tglobal#1{\global\let#1#1}

```

The second one. We need to patch \uclclist, but it is done once and only if \SetCase is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact \uclclist is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually \reserved@a), we pass it as argument to \bbl@uclc. The parser is restarted inside \<lang>\bbl@uclc because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

```
\let\bbl@tolower\empty\bbl@toupper\empty
```

and starts over (and similarly when lowercasing).

```

1756 \@ifpackagewith{babel}{nocase}%
1757   {\let\bbl@patchuclc\relax}%
1758   {\def\bbl@patchuclc% TODO. Delete. Doesn’t work any more.
1759     \global\let\bbl@patchuclc\relax
1760     \g@addto@macro\uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1761     \gdef\bbl@uclc##1{%
1762       \let\bbl@encoded\bbl@encoded@uclc
1763       \bbl@ifunset{\language\name @bbl@uclc}% and resumes it
1764       {##1}%

```

```

1765      {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1766      \csname\language @bbl@uclc\endcsname}%
1767      {\bbl@tolower\@empty}\bbl@toupper\@empty}}%
1768      \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1769      \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}%
1770 <<(*More package options)>> ≡
1771 \DeclareOption{nocase}{}
1772 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1773 <<(*More package options)>> ≡
1774 \let\bbl@opt@strings\@nnil % accept strings=value
1775 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1776 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1777 \def\BabelStringsDefault{generic}
1778 <</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.

```

1779 \@onlypreamble\StartBabelCommands
1780 \def\StartBabelCommands{%
1781   \begingroup
1782   \@tempcnta="7F
1783   \def\bbl@tempa{%
1784     \ifnum\@tempcnta>"FF\else
1785       \catcode\@tempcnta=11
1786       \advance\@tempcnta\@ne
1787       \expandafter\bbl@tempa
1788     \fi}%
1789   \bbl@tempa
1790   <<Macros local to BabelCommands>>
1791   \def\bbl@provstring##1##2{%
1792     \providecommand##1{##2}%
1793     \bbl@tglobal##1}%
1794   \global\let\bbl@scafter\@empty
1795   \let\StartBabelCommands\bbl@startcmds
1796   \ifx\BabelLanguages\relax
1797     \let\BabelLanguages\CurrentOption
1798   \fi
1799   \begingroup
1800   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1801   \StartBabelCommands}
1802 \def\bbl@startcmds{%
1803   \ifx\bbl@screset\@nnil\else
1804     \bbl@usehooks{stopcommands}{}%
1805   \fi
1806   \endgroup
1807   \begingroup
1808   \@ifstar
1809     {\ifx\bbl@opt@strings\@nnil
1810       \let\bbl@opt@strings\BabelStringsDefault
1811     \fi
1812     \bbl@startcmds@i}%
1813   \bbl@startcmds@i}
1814 \def\bbl@startcmds@i#1#2{%
1815   \edef\bbl@L{\zap@space#1 \@empty}%
1816   \edef\bbl@G{\zap@space#2 \@empty}%
1817   \bbl@startcmds@ii}
1818 \let\bbl@startcommands\StartBabelCommands

```

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

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

```

1819 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1820   \let\SetString\@gobbletwo
1821   \let\bbl@stringdef\@gobbletwo
1822   \let\AfterBabelCommands\@gobble
1823   \ifx\@empty#1%
1824     \def\bbl@sc@label{generic}%
1825     \def\bbl@encstring##1##2{%
1826       \ProvideTextCommandDefault##1{##2}%
1827       \bbl@toglobal##1%
1828       \expandafter\bbl@toglobal\curname\string?\string##1\endcsname}%
1829     \let\bbl@sctest\in@true
1830   \else
1831     \let\bbl@sc@charset\space % <- zapped below
1832     \let\bbl@sc@fontenc\space % <- " "
1833     \def\bbl@tempa##1=##2\@nil{%
1834       \bbl@csarg\edef{sc@zap@space##1 \@empty}{##2 }}%
1835     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1836     \def\bbl@tempa##1 ##2{% space -> comma
1837       ##1%
1838       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1839     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1840     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1841     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1842     \def\bbl@encstring##1##2{%
1843       \bbl@foreach\bbl@sc@fontenc{%
1844         \bbl@ifunset{T@###1}%
1845         {}%
1846         {\ProvideTextCommand##1{###1}{##2}%
1847          \bbl@toglobal##1%
1848          \expandafter
1849          \bbl@toglobal\curname###1\string##1\endcsname}}}%
1850     \def\bbl@sctest{%
1851       \bbl@xin@{\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1852   \fi
1853   \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1854   \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1855     \let\AfterBabelCommands\bbl@aftercmds
1856     \let\SetString\bbl@setstring
1857     \let\bbl@stringdef\bbl@encstring
1858   \else % ie, strings=value
1859     \bbl@sctest
1860   \fin@
1861     \let\AfterBabelCommands\bbl@aftercmds
1862     \let\SetString\bbl@setstring
1863     \let\bbl@stringdef\bbl@provstring
1864   \fi\fi\fi
1865   \bbl@scswitch
1866   \ifx\bbl@G\@empty
1867     \def\SetString##1##2{%
1868       \bbl@error{Missing group for string \string##1}%
1869       {You must assign strings to some category, typically\\%
1870        captions or extras, but you set none}}%
1871   \fi
1872   \ifx\@empty#1%
1873     \bbl@usehooks{defaultcommands}{}%

```

```

1874 \else
1875   \@expandtwoargs
1876   \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1877 \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).

```

1878 \def\bbl@forlang#1#2{%
1879   \bbl@for#1\bbl@L{%
1880     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1881     \ifin@#2\relax\fi}}
1882 \def\bbl@scswitch{%
1883   \bbl@forlang\bbl@tempa{%
1884     \ifx\bbl@G\@empty\else
1885       \ifx\SetString\@gobbletwo\else
1886         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1887         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1888         \ifin@\else
1889           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1890           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1891         \fi
1892       \fi
1893     \fi}}
1894 \AtEndOfPackage{%
1895   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{\#2}}}%
1896   \let\bbl@scswitch\relax}
1897 \onlypreamble\EndBabelCommands
1898 \def\EndBabelCommands{%
1899   \bbl@usehooks{stopcommands}{}%
1900   \endgroup
1901   \endgroup
1902   \bbl@scafter}
1903 \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.

```

1904 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1905   \bbl@forlang\bbl@tempa{%
1906     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1907     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1908     {\bbl@exp{%
1909       \global\\ \bbl@add\<\bbl@G\bbl@tempa>{\ \bbl@scset\#1\<\bbl@LC>}}}%
1910     }%
1911     \def\BabelString{#2}%
1912     \bbl@usehooks{stringprocess}{}%
1913     \expandafter\bbl@stringdef
1914     \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}

```

Now, some additional stuff to be used when encoded strings are used. Captions then include `\bbl@encoded` for string to be expanded in case transformations. It is `\relax` by default, but in `\MakeUppercase` and `\MakeLowercase` its value is a modified expandable `\@changed@cmd`.

```

1915 \ifx\bbl@opt@strings\relax

```

```

1916 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1917 \bbl@patchuclc
1918 \let\bbl@encoded\relax
1919 \def\bbl@encoded@uclc#1{%
1920   \@inmathwarn#1%
1921   \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1922     \expandafter\ifx\csname ?\string#1\endcsname\relax
1923       \TextSymbolUnavailable#1%
1924     \else
1925       \csname ?\string#1\endcsname
1926     \fi
1927   \else
1928     \csname\cf@encoding\string#1\endcsname
1929   \fi}
1930 \else
1931   \def\bbl@scset#1#2{\def#1{#2}}
1932 \fi

```

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.

```

1933 <<*Macros local to BabelCommands>> ≡
1934 \def\SetStringLoop##1##2{%
1935   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1936   \count@\z@
1937   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1938     \advance\count@\@ne
1939     \toks@\expandafter{\bbl@tempa}%
1940     \bbl@exp{%
1941       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1942       \count@=\the\count@\relax}}}%
1943 <</Macros local to BabelCommands>>

```

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

```

1944 \def\bbl@aftercmds#1{%
1945   \toks@\expandafter{\bbl@scafter#1}%
1946   \xdef\bbl@scafter{\the\toks@}

```

Case mapping The command `\SetCase` provides a way to change the behavior of `\MakeUppercase` and `\MakeLowercase`. `\bbl@tempa` is set by the patched `\@uclclist` to the parsing command. *Deprecated*.

```

1947 <<*Macros local to BabelCommands>> ≡
1948 \newcommand\SetCase[3][{%
1949   \bbl@patchuclc
1950   \bbl@forlang\bbl@tempa{%
1951     \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1952     \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1953     \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1954 <</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.

```

1955 <<*Macros local to BabelCommands>> ≡
1956 \newcommand\SetHyphenMap[1]{%
1957   \bbl@forlang\bbl@tempa{%
1958     \expandafter\bbl@stringdef
1959     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1960 <</Macros local to BabelCommands>>

```

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

```

1961 \newcommand\BabelLower[2]{% one to one.

```



```

1962 \ifnum\lccode#1=#2\else
1963   \babel@savevariable{\lccode#1}%
1964   \lccode#1=#2\relax
1965 \fi}
1966 \newcommand\BabelLowerMM[4]{% many-to-many
1967   \@tempcnta=#1\relax
1968   \@tempcntb=#4\relax
1969   \def\bbl@tempa{%
1970     \ifnum\@tempcnta>#2\else
1971       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1972       \advance\@tempcnta#3\relax
1973       \advance\@tempcntb#3\relax
1974       \expandafter\bbl@tempa
1975     \fi}%
1976   \bbl@tempa}
1977 \newcommand\BabelLowerM0[4]{% many-to-one
1978   \@tempcnta=#1\relax
1979   \def\bbl@tempa{%
1980     \ifnum\@tempcnta>#2\else
1981       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1982       \advance\@tempcnta#3
1983       \expandafter\bbl@tempa
1984     \fi}%
1985   \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1986 <<{*More package options}> \equiv
1987 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1988 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1989 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1990 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1991 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1992 <</More package options>>

```

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

```

1993 \AtEndOfPackage{%
1994   \ifx\bbl@opt@hyphenmap\undefined
1995     \bbl@xin@{,}{\bbl@language@opts}%
1996     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1997   \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.

```

1998 \newcommand\setlocalecaption{% TODO. Catch typos.
1999   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2000 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2001   \bbl@trim@def\bbl@tempa{#2}%
2002   \bbl@xin@{.template}{\bbl@tempa}%
2003   \ifin@
2004     \bbl@ini@captions@template{#3}{#1}%
2005   \else
2006     \edef\bbl@tempd{%
2007       \expandafter\expandafter\expandafter
2008       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2009     \bbl@xin@
2010       {\expandafter\string\csname #2name\endcsname}%
2011       {\bbl@tempd}%
2012     \ifin@ % Renew caption
2013       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2014     \ifin@
2015       \bbl@exp{%
2016         \\bbl@ifsamestring{\bbl@tempa}{\language@name}%
2017         {\bbl@scset\<#2name>\<#1#2name>}}%

```

```

2018         {}}%
2019     \else % Old way converts to new way
2020         \bbl@ifunset{#1#2name}%
2021         {\bbl@exp{%
2022             \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
2023             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2024             {\def\<#2name>{\<#1#2name>}}}%
2025         {}}%
2026     }%
2027     \fi
2028     \else
2029         \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2030         \ifin@ % New way
2031             \bbl@exp{%
2032                 \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}}%
2033                 \\bbl@ifsamestring{\bbl@tempa}{\language}%
2034                 {\bbl@scset\<#2name>\<#1#2name>}}%
2035             {}}%
2036         \else % Old way, but defined in the new way
2037             \bbl@exp{%
2038                 \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
2039                 \\bbl@ifsamestring{\bbl@tempa}{\language}%
2040                 {\def\<#2name>{\<#1#2name>}}}%
2041             {}}%
2042         \fi%
2043     \fi
2044     \@namedef{#1#2name}{#3}%
2045     \toks@{\expandafter{\bbl@captionslist}}%
2046     \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2047     \ifin@ \else
2048         \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2049         \bbl@toglobal\bbl@captionslist
2050     \fi
2051 \fi}
2052 % \def\bbl@setcaption@s#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.

```

2053 \bbl@trace{Macros related to glyphs}
2054 \def\set@low@box#1{\setbox\tw@ \hbox{,}\setbox\z@ \hbox{#1}%
2055     \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2056     \setbox\z@ \hbox{\lower\dimen\z@ \box\z@}\ht\z@ \ht\tw@ \dp\z@ \dp\tw@}

```

`\save@s@f@q` The macro `\save@s@f@q` is used to save and reset the current space factor.

```

2057 \def\save@s@f@q#1{\leavevmode
2058     \begingroup
2059     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2060     \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.

```

2061 \ProvideTextCommand{\quotedblbase}{OT1}{%

```

```

2062 \save@sf@q{\set@low@box{\textquotedblright\}}%
2063 \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.

```

2064 \ProvideTextCommandDefault{\quotedblbase}{%
2065 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2066 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2067 \save@sf@q{\set@low@box{\textquoteright\}}%
2068 \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.

```

2069 \ProvideTextCommandDefault{\quotesinglbase}{%
2070 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2071 \ProvideTextCommand{\guillemetleft}{OT1}{%
2072 \ifmmode
2073 \ll
2074 \else
2075 \save@sf@q{\nobreak
2076 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2077 \fi}
2078 \ProvideTextCommand{\guillemetright}{OT1}{%
2079 \ifmmode
2080 \gg
2081 \else
2082 \save@sf@q{\nobreak
2083 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2084 \fi}
2085 \ProvideTextCommand{\guillemotleft}{OT1}{%
2086 \ifmmode
2087 \ll
2088 \else
2089 \save@sf@q{\nobreak
2090 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2091 \fi}
2092 \ProvideTextCommand{\guillemotright}{OT1}{%
2093 \ifmmode
2094 \gg
2095 \else
2096 \save@sf@q{\nobreak
2097 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2098 \fi}

```

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

```

2099 \ProvideTextCommandDefault{\guillemetleft}{%
2100 \UseTextSymbol{OT1}{\guillemetleft}}
2101 \ProvideTextCommandDefault{\guillemetright}{%
2102 \UseTextSymbol{OT1}{\guillemetright}}
2103 \ProvideTextCommandDefault{\guillemotleft}{%
2104 \UseTextSymbol{OT1}{\guillemotleft}}
2105 \ProvideTextCommandDefault{\guillemotright}{%
2106 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2107 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2108 \ifmmode
2109 <%
2110 \else
2111 \save@sf@q{\nobreak

```

```

2112 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2113 \fi}
2114 \ProvideTextCommand{\guilsinglright}{OT1}{%
2115 \ifmmode
2116 >%
2117 \else
2118 \save@sf@q{\nobreak
2119 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2120 \fi}

```

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

```

2121 \ProvideTextCommandDefault{\guilsinglleft}{%
2122 \UseTextSymbol{OT1}{\guilsinglleft}}
2123 \ProvideTextCommandDefault{\guilsinglright}{%
2124 \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.

```

2125 \DeclareTextCommand{\ij}{OT1}{%
2126 i\kern-0.02em\bbl@allowhyphens j}
2127 \DeclareTextCommand{\IJ}{OT1}{%
2128 I\kern-0.02em\bbl@allowhyphens J}
2129 \DeclareTextCommand{\ij}{T1}{\char188}
2130 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2131 \ProvideTextCommandDefault{\ij}{%
2132 \UseTextSymbol{OT1}{\ij}}
2133 \ProvideTextCommandDefault{\IJ}{%
2134 \UseTextSymbol{OT1}{\IJ}}

```

\dj \DJ The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```

2135 \def\crrtic@{\hrule height0.1ex width0.3em}
2136 \def\crttic@{\hrule height0.1ex width0.33em}
2137 \def\ddj@{%
2138 \setbox0\hbox{d}\dimen@=\ht0
2139 \advance\dimen@lex
2140 \dimen@.45\dimen@
2141 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2142 \advance\dimen@ii.5ex
2143 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2144 \def\DDJ@{%
2145 \setbox0\hbox{D}\dimen@=.55\ht0
2146 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2147 \advance\dimen@ii.15ex % correction for the dash position
2148 \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2149 \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2150 \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2151 %
2152 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2153 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2154 \ProvideTextCommandDefault{\dj}{%
2155 \UseTextSymbol{OT1}{\dj}}
2156 \ProvideTextCommandDefault{\DJ}{%
2157 \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.

```
2158 \DeclareTextCommand{\SS}{OT1}{SS}
2159 \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
2160 \ProvideTextCommandDefault{\glq}{%
2161   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2162 \ProvideTextCommand{\grq}{T1}{%
2163   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}}
2164 \ProvideTextCommand{\grq}{TU}{%
2165   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}}
2166 \ProvideTextCommand{\grq}{OT1}{%
2167   \save@sf@q{\kern-.0125em
2168     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2169     \kern.07em\relax}}
2170 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}
```

`\glqq` The ‘german’ double quotes.

```
\grqq
2171 \ProvideTextCommandDefault{\glqq}{%
2172   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2173 \ProvideTextCommand{\grqq}{T1}{%
2174   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}}
2175 \ProvideTextCommand{\grqq}{TU}{%
2176   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}}
2177 \ProvideTextCommand{\grqq}{OT1}{%
2178   \save@sf@q{\kern-.07em
2179     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2180     \kern.07em\relax}}
2181 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}
```

`\flq` The ‘french’ single guillemets.

```
\frq
2182 \ProvideTextCommandDefault{\flq}{%
2183   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}}
2184 \ProvideTextCommandDefault{\frq}{%
2185   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq
2186 \ProvideTextCommandDefault{\flqq}{%
2187   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}}
2188 \ProvideTextCommandDefault{\frqq}{%
2189   \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).

```
2190 \def\umlauthigh{%
2191   \def\bbl@umlauta##1{\leavevmode\bgroup%
2192     \accent\csname\fontencoding dpos\endcsname
2193     ##1\bbl@allowhyphens\egroup}%
2194   \let\bbl@umlaute\bbl@umlauta}
2195 \def\umlautlow{%
2196   \def\bbl@umlauta{\protect\lower@umlaut}}
2197 \def\umlautelow{%
2198   \def\bbl@umlaute{\protect\lower@umlaut}}
2199 \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.

```
2200 \expandafter\ifx\csname U@D\endcsname\relax
2201   \csname newdimen\endcsname\U@D
2202 \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.

```
2203 \def\lower@umlaut#1{%
2204   \leavevmode\bgroup
2205   \U@D lex%
2206   {\setbox\z@\hbox{%
2207     \char\csname\fontencoding dpos\endcsname}%
2208     \dimen@ -.45ex\advance\dimen@\ht\z@
2209     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2210   \accent\csname\fontencoding dpos\endcsname
2211   \fontdimen5\font\U@D #1%
2212   \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).

```
2213 \AtBeginDocument{%
2214   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2215   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2216   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2217   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2218   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2219   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2220   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2221   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2222   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2223   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2224   \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.

```
2225 \ifx\l@english\undefined
2226   \chardef\l@english\z@
2227 \fi
2228 % The following is used to cancel rules in ini files (see Amharic).
```

```

2229 \ifx\l@unhyphenated\@undefined
2230 \newlanguage\l@unhyphenated
2231 \fi

```

4.13 Layout

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

```

2232 \bbl@trace{Bidi layout}
2233 \providecommand\IfBabelLayout[3]{#3}%
2234 \core
2235 \newcommand\BabelPatchSection[1]{%
2236   \@ifundefined{#1}{}{%
2237     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2238     \@namedef{#1}{%
2239       \ifstar{\bbl@presec@#1}%
2240       {\@dblarg{\bbl@presec@#1}}}%
2241 \def\bbl@presec@x#1[#2]#3{%
2242   \bbl@exp{%
2243     \\\select@language@x{\bbl@main@language}%
2244     \\\bbl@cs{sspre@#1}%
2245     \\\bbl@cs{ss@#1}%
2246     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2247     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2248     \\\select@language@x{\language}}%
2249 \def\bbl@presec@s#1#2{%
2250   \bbl@exp{%
2251     \\\select@language@x{\bbl@main@language}%
2252     \\\bbl@cs{sspre@#1}%
2253     \\\bbl@cs{ss@#1}*%
2254     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2255     \\\select@language@x{\language}}%
2256 \IfBabelLayout{sectioning}%
2257   {\BabelPatchSection{part}%
2258    \BabelPatchSection{chapter}%
2259    \BabelPatchSection{section}%
2260    \BabelPatchSection{subsection}%
2261    \BabelPatchSection{subsubsection}%
2262    \BabelPatchSection{paragraph}%
2263    \BabelPatchSection{subparagraph}%
2264    \def\babel@toc#1{%
2265      \select@language@x{\bbl@main@language}}}%
2266 \IfBabelLayout{captions}%
2267   {\BabelPatchSection{caption}}}%
2268 \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.

```

2269 \bbl@trace{Input engine specific macros}
2270 \ifcase\bbl@engine
2271   \input txtbabel.def
2272 \or
2273   \input luababel.def
2274 \or
2275   \input xebabel.def
2276 \fi
2277 \providecommand\babelfont{%
2278   \bbl@error
2279   {This macro is available only in LuaLaTeX and XeLaTeX.}%
2280   {Consider switching to these engines.}}
2281 \providecommand\babelprehyphenation{%

```

```

2282 \bbl@error
2283 {This macro is available only in LuaLaTeX.}%
2284 {Consider switching to that engine.}}
2285 \ifx\babelposthyphenation\undefined
2286 \let\babelposthyphenation\babelprehyphenation
2287 \let\babelpatterns\babelprehyphenation
2288 \let\babelcharproperty\babelprehyphenation
2289 \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.

```

2290 </package | core>
2291 <*package>
2292 \bbl@trace{Creating languages and reading ini files}
2293 \let\bbl@extend@ini@gobble
2294 \newcommand\babelprovide[2][]{%
2295   \let\bbl@savelangname\language
2296   \edef\bbl@savelocaleid{\the\localeid}%
2297   % Set name and locale id
2298   \edef\language{#2}%
2299   \bbl@id@assign
2300   % Initialize keys
2301   \bbl@vforeach{captions,date,import,main,script,language,%
2302     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2303     mapdigits,intraspaces,intrapenalty,onchar,transforms,alpha,%
2304     Alph,labels,labels*,calendar,date,casing}%
2305     {\bbl@csarg\let{KVP@##1}\@nnil}%
2306   \global\let\bbl@release@transforms\@empty
2307   \let\bbl@calendars\@empty
2308   \global\let\bbl@inidata\@empty
2309   \global\let\bbl@extend@ini@gobble
2310   \global\let\bbl@included@inis\@empty
2311   \gdef\bbl@key@list{;}%
2312   \bbl@forkv{#1}{%
2313     \in@{/}{##1}% With /, (re)sets a value in the ini
2314     \ifin@
2315       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2316       \bbl@renewinikey##1\@{##2}%
2317     \else
2318       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2319         \bbl@error
2320           {Unknown key '##1' in \string\babelprovide}%
2321           {See the manual for valid keys}%
2322       \fi
2323       \bbl@csarg\def{KVP@##1}{##2}%
2324     \fi}%
2325   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2326   \bbl@ifunset{date#2}\z@{\bbl@ifunset\bbl@llevel#2}\@ne\tw}%
2327   % == init ==
2328   \ifx\bbl@screaset\undefined
2329     \bbl@ldfinit
2330   \fi
2331   % == date (as option) ==
2332   % \ifx\bbl@KVP@date\@nnil\else
2333   % \fi
2334   % ==
2335   \let\bbl@lbfkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2336   \ifcase\bbl@howloaded
2337     \let\bbl@lbfkflag\@empty % new

```



```

2338 \else
2339   \ifx\bbk@KVP@hyphenrules\@nnil\else
2340     \let\bbk@lbkflag\@empty
2341   \fi
2342   \ifx\bbk@KVP@import\@nnil\else
2343     \let\bbk@lbkflag\@empty
2344   \fi
2345 \fi
2346 % == import, captions ==
2347 \ifx\bbk@KVP@import\@nnil\else
2348   \bbl@exp{\bbk@ifblank{\bbk@KVP@import}}%
2349   {\ifx\bbk@initload\relax
2350     \begingroup
2351       \def\BabelBeforeIni##1##2{\gdef\bbk@KVP@import{##1}\endinput}%
2352       \bbl@input@texini{##2}%
2353     \endgroup
2354   \else
2355     \xdef\bbk@KVP@import{\bbk@initload}%
2356   \fi}%
2357 {}%
2358 \let\bbk@KVP@date\@empty
2359 \fi
2360 \let\bbk@KVP@captions@\bbk@KVP@captions % TODO. A dirty hack
2361 \ifx\bbk@KVP@captions\@nnil
2362   \let\bbk@KVP@captions\bbk@KVP@import
2363 \fi
2364 % ==
2365 \ifx\bbk@KVP@transforms\@nnil\else
2366   \bbl@replace\bbk@KVP@transforms{ }{,}%
2367 \fi
2368 % == Load ini ==
2369 \ifcase\bbk@howloaded
2370   \bbl@provide@new{##2}%
2371 \else
2372   \bbl@ifblank{##1}%
2373   {}% With \bbl@load@basic below
2374   {\bbl@provide@renew{##2}}%
2375 \fi
2376 % == include == TODO
2377 % \ifx\bbk@included@inis\@empty\else
2378 %   \bbl@replace\bbk@included@inis{ }{,}%
2379 %   \bbl@foreach\bbk@included@inis{%
2380 %     \openin\bbk@readstream=babel-##1.ini
2381 %     \bbl@extend@ini{##2}%
2382 %     \closein\bbk@readstream
2383 %   \fi
2384 % Post tasks
2385 % -----
2386 % == subsequent calls after the first provide for a locale ==
2387 \ifx\bbk@inidata\@empty\else
2388   \bbl@extend@ini{##2}%
2389 \fi
2390 % == ensure captions ==
2391 \ifx\bbk@KVP@captions\@nnil\else
2392   \bbl@ifunset{\bbk@extracaps@##2}%
2393   {\bbl@exp{\bbk@babelensure[exclude=\bbk@today]{##2}}}%
2394   {\bbl@exp{\bbk@babelensure[exclude=\bbk@today,
2395     include=\bbk@extracaps@##2]{##2}}}%
2396   \bbl@ifunset{\bbk@ensure@\languagename}%
2397   {\bbl@exp{%
2398     \\\DeclareRobustCommand\<\bbk@ensure@\languagename>[1]{%
2399       \\\foreignlanguage{\languagename}%
2400       {###1}}}%

```

```

2401     {}%
2402     \bbl@exp{%
2403         \\bbl@tglobal\<bbl@ensure@\language\name>%
2404         \\bbl@tglobal\<bbl@ensure@\language\name\space>%
2405     \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.

```

2406     \bbl@load@basic{#2}%
2407     % == script, language ==
2408     % Override the values from ini or defines them
2409     \ifx\bbl@KVP@script\@nnil\else
2410         \bbl@csarg\edef{sname#2}{\bbl@KVP@script}%
2411     \fi
2412     \ifx\bbl@KVP@language\@nnil\else
2413         \bbl@csarg\edef{lname#2}{\bbl@KVP@language}%
2414     \fi
2415     \ifcase\bbl@engine\or
2416         \bbl@ifunset\bbl@chrng@\language\name{}\}%
2417         {\directlua{
2418             Babel.set_chranges_b('\bbl@cl{sbc}', '\bbl@cl{chrng}') }}%
2419     \fi
2420     % == onchar ==
2421     \ifx\bbl@KVP@onchar\@nnil\else
2422         \bbl@lua\hyphenate
2423         \bbl@exp{%
2424             \\AddToHook{env/document/before}{\select@language{#2}}}%
2425         \directlua{
2426             if Babel.locale_mapped == nil then
2427                 Babel.locale_mapped = true
2428                 Babel.linebreaking.add_before(Babel.locale_map, 1)
2429                 Babel.loc_to_scr = {}
2430                 Babel.chr_to_loc = Babel.chr_to_loc or {}
2431             end
2432             Babel.locale_props[\the\localeid].letters = false
2433         }%
2434         \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2435     \ifin@
2436         \directlua{
2437             Babel.locale_props[\the\localeid].letters = true
2438         }%
2439     \fi
2440     \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2441     \ifin@
2442         \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2443             \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2444         \fi
2445         \bbl@exp{\\bbl@add\\bbl@starthyphens
2446             {\bbl@patterns@lua{\language\name}}}%
2447         % TODO - error/warning if no script
2448         \directlua{
2449             if Babel.script_blocks['\bbl@cl{sbc}'] then
2450                 Babel.loc_to_scr[\the\localeid] =
2451                     Babel.script_blocks['\bbl@cl{sbc}']
2452                 Babel.locale_props[\the\localeid].lc = \the\localeid\space
2453                 Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language\name}\space
2454             end
2455         }%
2456     \fi
2457     \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2458     \ifin@
2459         \bbl@ifunset\bbl@lsys@\language\name{\bbl@provide@lsys{\language\name}}}%

```

```

2460 \bbl@ifunset{\bbl@wdir\language}\bbl@provide@dirs{\language}}{}%
2461 \directlua{
2462   if Babel.script_blocks['\bbl@cl{sbc}'] then
2463     Babel.loc_to_scr[\the\localeid] =
2464       Babel.script_blocks['\bbl@cl{sbc}']
2465   end}%
2466 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2467 \AtBeginDocument{%
2468   \bbl@patchfont{\bbl@mapselect}%
2469   {\selectfont}%
2470 \def\bbl@mapselect{%
2471   \let\bbl@mapselect\relax
2472   \edef\bbl@prefontid{\fontid\font}%
2473 \def\bbl@mapdir##1{%
2474   {\def\language{##1}%
2475   \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2476   \bbl@switchfont
2477   \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2478     \directlua{
2479       Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2480       [\bbl@prefontid] = \fontid\font\space}%
2481     \fi}}%
2482 \fi
2483 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2484 \fi
2485 % TODO - catch non-valid values
2486 \fi
2487 % == mapfont ==
2488 % For bidi texts, to switch the font based on direction
2489 \ifx\bbl@KVP@mapfont\@nnil\else
2490 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2491 {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\%
2492   mapfont. Use 'direction'.%
2493   {See the manual for details.}}}%
2494 \bbl@ifunset{\bbl@sys\language}\bbl@provide@sys{\language}}{}%
2495 \bbl@ifunset{\bbl@wdir\language}\bbl@provide@dirs{\language}}{}%
2496 \ifx\bbl@mapselect\undefined % TODO. See onchar.
2497 \AtBeginDocument{%
2498   \bbl@patchfont{\bbl@mapselect}%
2499   {\selectfont}%
2500 \def\bbl@mapselect{%
2501   \let\bbl@mapselect\relax
2502   \edef\bbl@prefontid{\fontid\font}%
2503 \def\bbl@mapdir##1{%
2504   {\def\language{##1}%
2505   \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2506   \bbl@switchfont
2507   \directlua{Babel.fontmap
2508     [\the\csname bbl@wdir##1\endcsname]%
2509     [\bbl@prefontid]=\fontid\font}}}%
2510 \fi
2511 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
2512 \fi
2513 % == Line breaking: intraspace, intrapenalty ==
2514 % For CJK, East Asian, Southeast Asian, if interspace in ini
2515 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2516 \bbl@csarg\edef{intsp#2}{\bbl@KVP@intraspace}%
2517 \fi
2518 \bbl@provide@intraspace
2519 % == Line breaking: CJK quotes == TODO -> @extras
2520 \ifcase\bbl@engine\or
2521 \bbl@xin{/c}{\bbl@cl{lnbrk}}%
2522 \ifin@

```

```

2523 \bbl@ifunset{bbl@quote@\languagename}{}%
2524 {\directlua{
2525   Babel.locale_props[\the\localeid].cjk_quotes = {}
2526   local cs = 'op'
2527   for c in string.utfvalues(
2528     [[\csname bbl@quote@\languagename\endcsname]]) do
2529     if Babel.cjk_characters[c].c == 'qu' then
2530       Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2531     end
2532     cs = ( cs == 'op') and 'cl' or 'op'
2533   end
2534 }}%
2535 \fi
2536 \fi
2537 % == Line breaking: justification ==
2538 \ifx\bbl@KVP@justification\@nnil\else
2539   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2540 \fi
2541 \ifx\bbl@KVP@linebreaking\@nnil\else
2542   \bbl@xin@{,\bbl@KVP@linebreaking,}%
2543   {,elongated,kashida,cjk,padding,unhyphenated,}%
2544   \ifin@
2545     \bbl@csarg\xdef
2546     {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2547   \fi
2548 \fi
2549 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
2550 \ifin@else\bbl@xin@{/k}{/\bbl@cl{\lnbrk}}\fi
2551 \ifin@\bbl@arabicjust\fi
2552 \bbl@xin@{/p}{/\bbl@cl{\lnbrk}}%
2553 \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2554 % == Line breaking: hyphenate.other.(locale|script) ==
2555 \ifx\bbl@lbfkflag\@empty
2556   \bbl@ifunset{bbl@hyotl@\languagename}{}%
2557   {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}}%
2558   \bbl@startcommands*\languagename}{}%
2559   \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2560     \ifcase\bbl@engine
2561       \ifnum##1<257
2562         \SetHyphenMap{\BabelLower{##1}{##1}}%
2563       \fi
2564     \else
2565       \SetHyphenMap{\BabelLower{##1}{##1}}%
2566     \fi}%
2567   \bbl@endcommands}%
2568 \bbl@ifunset{bbl@hyots@\languagename}{}%
2569 {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}}%
2570 \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2571   \ifcase\bbl@engine
2572     \ifnum##1<257
2573       \global\lccode##1=##1\relax
2574     \fi
2575   \else
2576     \global\lccode##1=##1\relax
2577   \fi}}%
2578 \fi
2579 % == Counters: maparabic ==
2580 % Native digits, if provided in ini (TeX level, xe and lua)
2581 \ifcase\bbl@engine\else
2582   \bbl@ifunset{bbl@dgnat@\languagename}{}%
2583   {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2584     \expandafter\expandafter\expandafter
2585     \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname

```

```

2586 \ifx\bbbl@KVP@maparabic\@nnil\else
2587 \ifx\bbbl@latinarabic\@undefined
2588 \expandafter\let\expandafter\@arabic
2589 \csname bbl@counter@languagename\endcsname
2590 \else % ie, if layout=counters, which redefines \@arabic
2591 \expandafter\let\expandafter\bbbl@latinarabic
2592 \csname bbl@counter@languagename\endcsname
2593 \fi
2594 \fi
2595 \fi}%
2596 \fi
2597 % == Counters: mapdigits ==
2598 % > luababel.def
2599 % == Counters: alph, Alph ==
2600 \ifx\bbbl@KVP@alph\@nnil\else
2601 \bbbl@exp{%
2602 \\\bbbl@add\<bbbl@preextras@languagename>{%
2603 \\\babel@save\\\@alph
2604 \let\\\@alph\<bbbl@cntr@\bbbl@KVP@alph @languagename>}}%
2605 \fi
2606 \ifx\bbbl@KVP@Alph\@nnil\else
2607 \bbbl@exp{%
2608 \\\bbbl@add\<bbbl@preextras@languagename>{%
2609 \\\babel@save\\\@Alph
2610 \let\\\@Alph\<bbbl@cntr@\bbbl@KVP@Alph @languagename>}}%
2611 \fi
2612 % == Casing ==
2613 \ifx\bbbl@KVP@casing\@nnil\else
2614 \bbbl@csarg\xdef{casing@languagename}%
2615 {\@nameuse{bbbl@casing@languagename}-x-\bbbl@KVP@casing}%
2616 \fi
2617 % == Calendars ==
2618 \ifx\bbbl@KVP@calendar\@nnil
2619 \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}%
2620 \fi
2621 \def\bbbl@tempe##1 ##2\@{ % Get first calendar
2622 \def\bbbl@tempa{##1}}%
2623 \bbbl@exp{\\\bbbl@tempe\bbbl@KVP@calendar\space\\\@}%
2624 \def\bbbl@tempe##1.##2.##3\@{
2625 \def\bbbl@tempc{##1}%
2626 \def\bbbl@tempb{##2}}%
2627 \expandafter\bbbl@tempe\bbbl@tempa.\@
2628 \bbbl@csarg\edef{calpr@languagename}{%
2629 \ifx\bbbl@tempc@empty\else
2630 calendar=\bbbl@tempc
2631 \fi
2632 \ifx\bbbl@tempb@empty\else
2633 ,variant=\bbbl@tempb
2634 \fi}%
2635 % == engine specific extensions ==
2636 % Defined in XXXbabel.def
2637 \bbbl@provide@extra{#2}%
2638 % == require.babel in ini ==
2639 % To load or reload the babel-*.tex, if require.babel in ini
2640 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2641 \bbbl@ifunset{bbbl@rtex@languagename}{%
2642 {\expandafter\ifx\csname bbl@rtex@languagename\endcsname\@empty\else
2643 \let\BabelBeforeIni\@gobbletwo
2644 \chardef\atcatcode=\catcode`\@
2645 \catcode`\@=11\relax
2646 \bbbl@input@texini{\bbbl@cs{rtex@languagename}}%
2647 \catcode`\@=\atcatcode
2648 \let\atcatcode\relax

```

```

2649     \global\bbbl@csarg\let{rqtex@}\language}\relax
2650     \fi}%
2651     \bbbl@foreach\bbbl@calendars{%
2652     \bbbl@ifunset{bbbl@ca##1}{%
2653     \chardef\atcatcode=\catcode`\@
2654     \catcode`\@=11\relax
2655     \InputIfFileExists{babel-ca-##1.tex}{}{}%
2656     \catcode`\@=\atcatcode
2657     \let\atcatcode\relax}%
2658     {}}%
2659     \fi
2660     % == frenchspacing ==
2661     \ifcasebbbl@howloaded\in@true\else\in@false\fi
2662     \ifin@ \else\bbbl@xin@{typography/frenchspacing}{\bbbl@key@list}\fi
2663     \ifin@
2664     \bbbl@extras@wrap{\\bbbl@pre@fs}%
2665     {\bbbl@pre@fs}%
2666     {\bbbl@post@fs}%
2667     \fi
2668     % == transforms ==
2669     % > luababel.def
2670     % == main ==
2671     \ifx\bbbl@KVP@main\@nnil % Restore only if not 'main'
2672     \let\language\bbbl@savelangname
2673     \chardef\localeid\bbbl@savelocaleid\relax
2674     \fi
2675     % == hyphenrules (apply if current) ==
2676     \ifx\bbbl@KVP@hyphenrules\@nnil\else
2677     \ifnum\bbbl@savelocaleid=\localeid
2678     \language\@nameuse{l@\language}\relax
2679     \fi
2680     \fi}

```

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

```

2681 \def\bbbl@provide@new#1{%
2682   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2683   \@namedef{extras#1}{}%
2684   \@namedef{noextras#1}{}%
2685   \bbbl@startcommands*{#1}{captions}%
2686   \ifx\bbbl@KVP@captions\@nnil % and also if import, implicit
2687   \def\bbbl@tempb##1% elt for \bbbl@captionslist
2688   \ifx##1\@empty\else
2689   \bbbl@exp{%
2690     \\SetString\\##1{%
2691       \\bbbl@nocaption{\bbbl@stripslash##1}{#1\bbbl@stripslash##1}}}%
2692   \expandafter\bbbl@tempb
2693   \fi}%
2694   \expandafter\bbbl@tempb\bbbl@captionslist\@empty
2695   \else
2696   \ifx\bbbl@initoload\relax
2697   \bbbl@read@ini{\bbbl@KVP@captions}2% % Here letters cat = 11
2698   \else
2699   \bbbl@read@ini{\bbbl@initoload}2% % Same
2700   \fi
2701   \fi
2702   \StartBabelCommands*{#1}{date}%
2703   \ifx\bbbl@KVP@date\@nnil
2704   \bbbl@exp{%
2705     \\SetString\\today{\bbbl@nocaption{today}{#1today}}}%
2706   \else
2707   \bbbl@savetoday
2708   \bbbl@savestate

```

```

2709 \fi
2710 \bbl@endcommands
2711 \bbl@load@basic{#1}%
2712 % == hyphenmins == (only if new)
2713 \bbl@exp{%
2714 \gdef\<#1hyphenmins>{%
2715 {\bbl@ifunset{\bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2716 {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}%
2717 % == hyphenrules (also in renew) ==
2718 \bbl@provide@hyphens{#1}%
2719 \ifx\bbl@KVP@main\@nnil\else
2720 \expandafter\main@language\expandafter{#1}%
2721 \fi}
2722 %
2723 \def\bbl@provide@renew#1{%
2724 \ifx\bbl@KVP@captions\@nnil\else
2725 \StartBabelCommands*{#1}{captions}%
2726 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
2727 \EndBabelCommands
2728 \fi
2729 \ifx\bbl@KVP@date\@nnil\else
2730 \StartBabelCommands*{#1}{date}%
2731 \bbl@savetoday
2732 \bbl@savedate
2733 \EndBabelCommands
2734 \fi
2735 % == hyphenrules (also in new) ==
2736 \ifx\bbl@lbkflag\@empty
2737 \bbl@provide@hyphens{#1}%
2738 \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.)

```

2739 \def\bbl@load@basic#1{%
2740 \ifcase\bbl@howloaded\or\or
2741 \ifcase\csname bbl@llevel@\language\endcsname
2742 \bbl@csarg\let{\lname@\language}\relax
2743 \fi
2744 \fi
2745 \bbl@ifunset{\bbl@lname@#1}%
2746 {\def\BabelBeforeIni##1##2{%
2747 \begingroup
2748 \let\bbl@ini@captions\aux\@gobbletwo
2749 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2750 \bbl@read@ini{##1}1%
2751 \ifx\bbl@initoload\relax\endinput\fi
2752 \endgroup}%
2753 \begingroup % boxed, to avoid extra spaces:
2754 \ifx\bbl@initoload\relax
2755 \bbl@input@texini{#1}%
2756 \else
2757 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2758 \fi
2759 \endgroup}%
2760 {}

```

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`.

```

2761 \def\bbl@provide@hyphens#1{%
2762 \@tempcnta\m@ne % a flag
2763 \ifx\bbl@KVP@hyphenrules\@nnil\else
2764 \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2765 \bbl@foreach\bbl@KVP@hyphenrules{%

```

```

2766 \ifnum\@tempcnta=\m@ne % if not yet found
2767 \bbl@ifsamestring{##1}{+}%
2768 {\bbl@carg\addlanguage{l@##1}}%
2769 {}%
2770 \bbl@ifunset{l@##1}% After a possible +
2771 {}%
2772 {\@tempcnta\@nameuse{l@##1}}%
2773 \fi}%
2774 \ifnum\@tempcnta=\m@ne
2775 \bbl@warning{%
2776 Requested 'hyphenrules' for '\language' not found:\%
2777 \bbl@KVP@hyphenrules.\%
2778 Using the default value. Reported}%
2779 \fi
2780 \fi
2781 \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2782 \ifx\bbl@KVP@captions@@\@nnil % TODO. Hackish. See above.
2783 \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2784 {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2785 {}%
2786 {\bbl@ifunset{l@bbl@cl{hyphr}}%
2787 {}% if hyphenrules found:
2788 {\@tempcnta\@nameuse{l@bbl@cl{hyphr}}}}%
2789 \fi
2790 \fi
2791 \bbl@ifunset{l@#1}%
2792 {\ifnum\@tempcnta=\m@ne
2793 \bbl@carg\adddialect{l@#1}\language
2794 \else
2795 \bbl@carg\adddialect{l@#1}\@tempcnta
2796 \fi}%
2797 {\ifnum\@tempcnta=\m@ne\else
2798 \global\bbl@carg\chardef{l@#1}\@tempcnta
2799 \fi}}

```

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

```

2800 \def\bbl@input@texini#1{%
2801 \bbl@bsphack
2802 \bbl@exp{%
2803 \catcode`\\%=14 \catcode`\\|=0
2804 \catcode`\\{=1 \catcode`\\|=2
2805 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
2806 \catcode`\\%=the\catcode`\% \relax
2807 \catcode`\\|=the\catcode`\| \relax
2808 \catcode`\\{=the\catcode`\{ \relax
2809 \catcode`\\|=the\catcode`\| \relax}%
2810 \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.

```

2811 \def\bbl@inline#1\bbl@inline{%
2812 \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1@@% ]
2813 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2814 \def\bbl@iniskip#1\@@{% if starts with ;
2815 \def\bbl@inistore#1=#2\@@{% full (default)
2816 \bbl@trim@def\bbl@tempa{#1}%
2817 \bbl@trim\toks{#2}%
2818 \bbl@xin@{\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2819 \ifin@else
2820 \bbl@xin@{,identification/include.}%
2821 {\bbl@section/\bbl@tempa}%
2822 \ifin@\xdef\bbl@included@inis{the\toks}\fi
2823 \bbl@exp{%

```



```

2824     \\g@addto@macro\\bbl@inidata{%
2825     \\bbl@elt{\\bbl@section}{\\bbl@tempa}{\\the\\toks@}}}%
2826 \fi}
2827 \def\bbl@inistore@min#1=#2\\@@{%  minimal (maybe set in \\bbl@read@ini)
2828 \bbl@trim@def\bbl@tempa{#1}%
2829 \bbl@trim\\toks@{#2}%
2830 \bbl@xin@{.identification.}{.\\bbl@section.}%
2831 \ifin@
2832 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
2833     \\bbl@elt{identification}{\\bbl@tempa}{\\the\\toks@}}}%
2834 \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.

```

2835 \def\bbl@loop@ini{%
2836 \loop
2837 \if T\\ifeof\bbl@readstream F\\fi T\\relax % Trick, because inside \\loop
2838 \endlinechar\\m@ne
2839 \read\bbl@readstream to \\bbl@line
2840 \endlinechar`\\^^M
2841 \ifx\\bbl@line\\empty\\else
2842 \expandafter\bbl@iniline\\bbl@line\\bbl@iniline
2843 \\fi
2844 \repeat}
2845 \ifx\\bbl@readstream\\undefined
2846 \csname newread\\endcsname\\bbl@readstream
2847 \\fi
2848 \def\bbl@read@ini#1#2{%
2849 \global\\let\\bbl@extend@ini\\gobble
2850 \openin\\bbl@readstream=babel-#1.ini
2851 \\ifeof\\bbl@readstream
2852 \bbl@error
2853 {There is no ini file for the requested language\\%
2854 (#1: \\language). Perhaps you misspelled it or your\\%
2855 installation is not complete.}%
2856 {Fix the name or reinstall babel.}%
2857 \\else
2858 % == Store ini data in \\bbl@inidata ==
2859 \catcode\\[=12 \catcode\\]=12 \catcode\\==12 \catcode\\&=12
2860 \catcode\\;=12 \catcode\\|=12 \catcode\\%=14 \catcode\\-=12
2861 \bbl@info{Importing
2862 \ifcase#2font and identification \\or basic \\fi
2863 data for \\language\\%
2864 from babel-#1.ini. Reported}%
2865 \ifnum#2=z@
2866 \global\\let\\bbl@inidata\\empty
2867 \\let\\bbl@inistore\\bbl@inistore@min % Remember it's local
2868 \\fi
2869 \def\bbl@section{identification}%
2870 \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2871 \bbl@inistore load.level=#2\\@@
2872 \bbl@loop@ini
2873 % == Process stored data ==
2874 \bbl@csarg\\xdef{\\lini@\\language}{#1}%
2875 \bbl@read@ini@aux
2876 % == 'Export' data ==
2877 \bbl@ini@exports{#2}%
2878 \global\\bbl@csarg\\let{\\inidata@\\language}\\bbl@inidata
2879 \\global\\let\\bbl@inidata\\empty

```

```

2880 \bbl@exp{\bbl@add@list\bbl@ini@loaded{\language\language}}%
2881 \bbl@tglobal\bbl@ini@loaded
2882 \fi
2883 \closein\bbl@readstream}
2884 \def\bbl@read@ini@aux{%
2885 \let\bbl@savestrings\empty
2886 \let\bbl@savetoday\empty
2887 \let\bbl@savestate\empty
2888 \def\bbl@elt##1##2##3{%
2889 \def\bbl@section{##1}%
2890 \in@{=date.}{=##1}% Find a better place
2891 \ifin@
2892 \bbl@ifunset{bbl@inikv@##1}%
2893 {\bbl@ini@calendar{##1}}%
2894 {}%
2895 \fi
2896 \bbl@ifunset{bbl@inikv@##1}{}%
2897 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2898 \bbl@inidata}

```

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

```

2899 \def\bbl@extend@ini@aux#1{%
2900 \bbl@startcommands*{#1}{captions}%
2901 % Activate captions/... and modify exports
2902 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2903 \setlocalecaption{#1}{##1}{##2}}%
2904 \def\bbl@inikv@captions##1##2{%
2905 \bbl@ini@captions@aux{##1}{##2}}%
2906 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2907 \def\bbl@exportkey##1##2##3{%
2908 \bbl@ifunset{bbl@kv@##2}{}%
2909 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2910 \bbl@exp{\global\let<bbl@##1@language>\<bbl@kv@##2>}}%
2911 \fi}}%
2912 % As with \bbl@read@ini, but with some changes
2913 \bbl@read@ini@aux
2914 \bbl@ini@exports\tw@
2915 % Update inidata@lang by pretending the ini is read.
2916 \def\bbl@elt##1##2##3{%
2917 \def\bbl@section{##1}%
2918 \bbl@iniline##2=##3\bbl@iniline}%
2919 \csname bbl@inidata@#1\endcsname
2920 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2921 \StartBabelCommands*{#1}{date}% And from the import stuff
2922 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2923 \bbl@savetoday
2924 \bbl@savestate
2925 \bbl@endcommands}

```

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

```

2926 \def\bbl@ini@calendar#1{%
2927 \lowercase{\def\bbl@tempa{=##1=}}%
2928 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2929 \bbl@replace\bbl@tempa{=date.}{}%
2930 \in@{.licr}{=##1=}%
2931 \ifin@
2932 \ifcase\bbl@engine
2933 \bbl@replace\bbl@tempa{.licr}{}%
2934 \else
2935 \let\bbl@tempa\relax
2936 \fi
2937 \fi
2938 \ifx\bbl@tempa\relax\else

```

```

2939 \bbl@replace\bbl@tempa{=}{}%
2940 \ifx\bbl@tempa\empty\else
2941 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2942 \fi
2943 \bbl@exp{%
2944 \def\<bbl@inikv@#1>####1####2{%
2945 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2946 \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).

```

2947 \def\bbl@renewinikey#1/#2\@#3{%
2948 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2949 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2950 \bbl@trim\toks@{#3}% value
2951 \bbl@exp{%
2952 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2953 \\\g@addto@macro\\bbl@inidata{%
2954 \\\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.

```

2955 \def\bbl@exportkey#1#2#3{%
2956 \bbl@ifunset{\bbl@kv@#2}%
2957 {\bbl@csarg\gdef{#1@\language}\{#3}}%
2958 {\expandafter\ifx\csname \bbl@kv@#2\endcsname\empty
2959 \bbl@csarg\gdef{#1@\language}\{#3}}%
2960 \else
2961 \bbl@exp{\global\let\<bbl@#1@\language>\<bbl@kv@#2>}%
2962 \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.

```

2963 \def\bbl@iniwarning#1{%
2964 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2965 {\bbl@warning{%
2966 From babel-\bbl@cs{lini@\language}.ini:\\%
2967 \bbl@cs{kv@identification.warning#1}\\%
2968 Reported }}}
2969 %
2970 \let\bbl@release@transforms\empty
2971 \def\bbl@ini@exports#1{%
2972 % Identification always exported
2973 \bbl@iniwarning}%
2974 \ifcase\bbl@engine
2975 \bbl@iniwarning{.pdf\latex}%
2976 \or
2977 \bbl@iniwarning{.lua\latex}%
2978 \or
2979 \bbl@iniwarning{.x\elatex}%
2980 \fi%
2981 \bbl@exportkey{llevel}{identification.load.level}{}%
2982 \bbl@exportkey{elname}{identification.name.english}{}%
2983 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2984 {\csname \bbl@elname@\language\endcsname}}%
2985 \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
2986 % Somewhat hackish. TODO
2987 \bbl@exportkey{casing}{identification.tag.bcp47}{}%

```

```

2988 \bbl@exportkey{lbcpr}{identification.language.tag.bcp47}{}%
2989 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2990 \bbl@exportkey{esname}{identification.script.name}{}%
2991 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
2992 {csname bbl@esname@language\endcsname}}%
2993 \bbl@exportkey{sbcpr}{identification.script.tag.bcp47}{}%
2994 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2995 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2996 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2997 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2998 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2999 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
3000 % Also maps bcp47 -> language
3001 \ifbbl@bcptoname
3002 \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpr}}{language}%
3003 \fi
3004 % Conditional
3005 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
3006 \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3007 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3008 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3009 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3010 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3011 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3012 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3013 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3014 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3015 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3016 \bbl@exportkey{chrng}{characters.ranges}{}%
3017 \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3018 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3019 \ifnum#1=\tw@ % only (re)new
3020 \bbl@exportkey{rqtex}{identification.require.babel}{}%
3021 \bbl@toglobal\bbl@savetoday
3022 \bbl@toglobal\bbl@savestate
3023 \bbl@savestrings
3024 \fi
3025 \fi}

```

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

```

3026 \def\bbl@inikv#1#2{% key=value
3027 \toks@{#2}% This hides #'s from ini values
3028 \bbl@csarg\edef{@kv@bbl@section.#1}{\the\toks@}}

```

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

```

3029 \let\bbl@inikv@identification\bbl@inikv
3030 \let\bbl@inikv@date\bbl@inikv
3031 \let\bbl@inikv@typography\bbl@inikv
3032 \let\bbl@inikv@characters\bbl@inikv
3033 \let\bbl@inikv@numbers\bbl@inikv

```

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’.

```

3034 \def\bbl@inikv@counters#1#2{%
3035 \bbl@ifsamestring{#1}{digits}%
3036 {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3037 decimal digits}%
3038 {Use another name.}}%
3039 }%
3040 \def\bbl@tempc#1{%
3041 \bbl@trim@def{\bbl@tempb*}{#2}%
3042 \in@{.1$}{#1$}%
3043 \ifin@

```

```

3044 \bbl@replace\bbl@tempc{.1}{}%
3045 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}\bbl@tempc}%
3046 \noexpand\bbl@alphanumeric{\bbl@tempc}%
3047 \fi
3048 \in@{.F.}{#1}%
3049 \ifin@else\in@{.S.}{#1}\fi
3050 \ifin@
3051 \bbl@csarg\protected@xdef{cntr@#1@\language}\bbl@tempb*}%
3052 \else
3053 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3054 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3055 \bbl@csarg\global\expandafter\let\{cntr@#1@\language}\bbl@tempa
3056 \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.

```

3057 \ifcase\bbl@engine
3058 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3059 \bbl@ini@captions@aux{#1}{#2}}
3060 \else
3061 \def\bbl@inikv@captions#1#2{%
3062 \bbl@ini@captions@aux{#1}{#2}}
3063 \fi

```

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

```

3064 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3065 \bbl@replace\bbl@tempa{.template}{}%
3066 \def\bbl@toreplace{#1}{}%
3067 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3068 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3069 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3070 \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}%
3071 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3072 \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3073 \ifin@
3074 \@nameuse{\bbl@patch\bbl@tempa}%
3075 \global\bbl@csarg\let\bbl@tempa fmt@#2\bbl@toreplace
3076 \fi
3077 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3078 \ifin@
3079 \global\bbl@csarg\let\bbl@tempa fmt@#2\bbl@toreplace
3080 \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3081 \\\bbl@ifunset\bbl@bbl@tempa fmt@\\language}%
3082 {\fnum@\bbl@tempa}}%
3083 {\\\@nameuse{\bbl@bbl@tempa fmt@\\language}}}%
3084 \fi}
3085 \def\bbl@ini@captions@aux#1#2{%
3086 \bbl@trim@def\bbl@tempa{#1}%
3087 \bbl@xin@{.template}{\bbl@tempa}%
3088 \ifin@
3089 \bbl@ini@captions@template{#2}\language
3090 \else
3091 \bbl@ifblank{#2}%
3092 {\bbl@exp{%
3093 \toks@{\\bbl@nocaption{\bbl@tempa}\language\bbl@tempa name}}}%
3094 {\bbl@trim\toks@{#2}}}%
3095 \bbl@exp{%
3096 \\\bbl@add\\bbl@savestrings{%
3097 \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3098 \toks@\expandafter{\bbl@captionslist}%
3099 \bbl@exp{\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3100 \ifin@else
3101 \bbl@exp{%

```

```

3102      \\bbl@add<bbl@extracaps@<language>{<bbl@tempa name>}%
3103      \\bbl@tglobal<bbl@extracaps@<language>}%
3104      \fi
3105      \fi}

```

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

```

3106 \def\bbl@list@the{%
3107   part,chapter,section,subsection,subsubsection,paragraph,%
3108   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3109   table,page,footnote,mpfootnote,mpfn}
3110 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3111   \bbl@ifunset{bbl@map@#1@<language>}%
3112     {\@nameuse{#1}}%
3113     {\@nameuse{bbl@map@#1@<language>}}}
3114 \def\bbl@inikv@labels#1#2{%
3115   \in@{.map}{#1}%
3116   \ifin@
3117     \ifx\bbl@KVP@labels\@nnil\else
3118       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3119       \ifin@
3120         \def\bbl@tempc{#1}%
3121         \bbl@replace\bbl@tempc{.map}{}%
3122         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3123         \bbl@exp{%
3124           \gdef<bbl@map@<bbl@tempc @<language>%
3125             {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3126         \bbl@foreach\bbl@list@the{%
3127           \bbl@ifunset{the##1}{}%
3128           {\bbl@exp{\let\\bbl@tempd<the##1>}%
3129             \bbl@exp{%
3130               \\bbl@sreplace<the##1>%
3131               {\<bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}%
3132               \\bbl@sreplace<the##1>%
3133               {\<@empty @<bbl@tempc><c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3134             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3135               \toks@ \expandafter\expandafter\expandafter{%
3136                 \csname the##1\endcsname}%
3137               \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3138             \fi}}%
3139         \fi
3140       \fi
3141     %
3142   \else
3143     %
3144     % The following code is still under study. You can test it and make
3145     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3146     % language dependent.
3147     \in@{enumerate.}{#1}%
3148     \ifin@
3149       \def\bbl@tempa{#1}%
3150       \bbl@replace\bbl@tempa{enumerate.}{}%
3151       \def\bbl@toreplace{#2}%
3152       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}%
3153       \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3154       \bbl@replace\bbl@toreplace{[ ]}{\endcsname}%
3155       \toks@ \expandafter{\bbl@toreplace}%
3156       % TODO. Execute only once:
3157       \bbl@exp{%
3158         \\bbl@add<extras<language>{%
3159           \\babel@save<labelenum\romannumeral\bbl@tempa>%
3160           \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3161         \\bbl@tglobal<extras<language>}%
3162       \fi

```

3163 \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.

```

3164 \def\bbl@chapttype{chapter}
3165 \ifx\@makechapterhead\undefined
3166   \let\bbl@patchchapter\relax
3167 \else\ifx\thechapter\undefined
3168   \let\bbl@patchchapter\relax
3169 \else\ifx\ps@headings\undefined
3170   \let\bbl@patchchapter\relax
3171 \else
3172   \def\bbl@patchchapter{%
3173     \global\let\bbl@patchchapter\relax
3174     \gdef\bbl@chfmt{%
3175       \bbl@ifunset{\bbl\bbl@chapttype fmt@\language}%
3176       {\@chapapp\space\thechapter}
3177       {\@nameuse{\bbl\bbl@chapttype fmt@\language}}}
3178     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3179     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3180     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3181     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3182     \bbl@tglobal\appendix
3183     \bbl@tglobal\ps@headings
3184     \bbl@tglobal\chaptermark
3185     \bbl@tglobal\@makechapterhead}
3186   \let\bbl@patchappendix\bbl@patchchapter
3187 \fi\fi\fi
3188 \ifx\@part\undefined
3189   \let\bbl@patchpart\relax
3190 \else
3191   \def\bbl@patchpart{%
3192     \global\let\bbl@patchpart\relax
3193     \gdef\bbl@partformat{%
3194       \bbl@ifunset{\bbl@partfmt@\language}%
3195       {\partname\nobreakspace\thepart}
3196       {\@nameuse{\bbl@partfmt@\language}}}
3197     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3198     \bbl@tglobal\@part}
3199 \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

```

3200 \let\bbl@calendar\empty
3201 \DeclareRobustCommand\localdate[1][\bbl@localdate{#1}]
3202 \def\bbl@localdate#1#2#3#4{%
3203   \begingroup
3204     \edef\bbl@they{#2}%
3205     \edef\bbl@them{#3}%
3206     \edef\bbl@thed{#4}%
3207     \edef\bbl@tempe{%
3208       \bbl@ifunset{\bbl@calpr@\language}{\bbl@cl{calpr}},%
3209       #1}%
3210     \bbl@replace\bbl@tempe{ }{}%
3211     \bbl@replace\bbl@tempe{CONVERT}{convert}% Hackish
3212     \bbl@replace\bbl@tempe{convert}{convert}%
3213     \let\bbl@ld@calendar\empty
3214     \let\bbl@ld@variant\empty
3215     \let\bbl@ld@convert\relax
3216     \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3217     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3218     \bbl@replace\bbl@ld@calendar{gregorian}{}%

```

```

3219 \ifx\babel@ld@calendar\@empty\else
3220 \ifx\babel@ld@convert\relax\else
3221 \babelcalendar[\babel@they-\babel@them-\babel@thed]%
3222 {\babel@ld@calendar}\babel@they\babel@them\babel@thed
3223 \fi
3224 \fi
3225 \@nameuse{\babel@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3226 \edef\babel@calendar{% Used in \month..., too
3227 \babel@ld@calendar
3228 \ifx\babel@ld@variant\@empty\else
3229 .\babel@ld@variant
3230 \fi}%
3231 \babel@cased
3232 {\@nameuse{\babel@date@\language name @\babel@calendar}%
3233 \babel@they\babel@them\babel@thed}%
3234 \endgroup}
3235 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3236 \def\babel@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3237 \babel@trim@def\babel@tempa{#1.#2}%
3238 \babel@ifsamestring{\babel@tempa}{months.wide}% to savedate
3239 {\babel@trim@def\babel@tempa{#3}%
3240 \babel@trim\toks@{#5}%
3241 \@temptokena\expandafter{\babel@savestate}%
3242 \babel@exp{% Reverse order - in ini last wins
3243 \def\babel@savestate{%
3244 \\\SetString<month\romannumeral\babel@tempa#6name>\the\toks@}%
3245 \the\@temptokena}}}%
3246 {\babel@ifsamestring{\babel@tempa}{date.long}% defined now
3247 {\lowercase{\def\babel@tempb{#6}}}%
3248 \babel@trim@def\babel@toreplace{#5}%
3249 \babel@TG@@date
3250 \global\babel@csarg\let{date@\language name @\babel@tempb}\babel@toreplace
3251 \ifx\babel@savestate\@empty
3252 \babel@exp{% TODO. Move to a better place.
3253 \\\AfterBabelCommands{%
3254 \def<\language name date>\\\protect<\language name date >%
3255 \\\newcommand<\language name date >[4][[]{%
3256 \\\babel@usedategroupttrue
3257 <\babel@ensure@\language name>{%
3258 \\\localedate[####1]{####2}{####3}{####4}}}%
3259 \def\babel@savestate{%
3260 \\\SetString\\today{%
3261 <\language name date>[convert]%
3262 {\the\year}{\the\month}{\the\day}}}%
3263 \fi}%
3264 {}}}}

```

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 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).

```

3265 \let\babel@calendar\@empty
3266 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3267 \@nameuse{\babel@ca#2}#1\@{
3268 \newcommand\BabelDateSpace{\nobreakspace}
3269 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3270 \newcommand\BabelDated[1]{\number#1}
3271 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3272 \newcommand\BabelDateM[1]{\ifnum#1<10 0\fi\number#1}
3273 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3274 \newcommand\BabelDateMMM[1]{\ifnum#1<10 0\fi\number#1}
3275 \csname month\romannumeral#1\babel@calendar name\endcsname}%

```



```

3276 \newcommand\BabelDatey[1]{\number#1}%
3277 \newcommand\BabelDateyy[1]{%
3278   \ifnum#1<10 0\number#1 %
3279   \else\ifnum#1<100 \number#1 %
3280   \else\ifnum#1<1000 \expandafter@gobble\number#1 %
3281   \else\ifnum#1<10000 \expandafter@gobbletwo\number#1 %
3282   \else
3283     \bbl@error
3284     {Currently two-digit years are restricted to the\
3285      range 0-9999.}%
3286     {There is little you can do. Sorry.}%
3287   \fi\fi\fi\fi}}
3288 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3289 \def\bbl@replace@finish@iii#1{%
3290   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3291 \def\bbl@TG@date{%
3292   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3293   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3294   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3295   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3296   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3297   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3298   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3299   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3300   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3301   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3302   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[###1|]}%
3303   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[###2|]}%
3304   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[###3|]}%
3305   \bbl@replace@finish@iii\bbl@toreplace}
3306 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3307 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3308 \let\bbl@release@transforms\empty
3309 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3310 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3311 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3312   #1[#2]{#3}{#4}{#5}}
3313 \begin{group} % A hack. TODO. Don't require an specific order
3314   \catcode\%=12
3315   \catcode\&=14
3316   \gdef\bbl@transforms#1#2#3{%&
3317     \directlua{
3318       local str = {[#2]==]
3319       str = str:gsub('%.%d+%.%d+$', '')
3320       token.set_macro('babeltempa', str)
3321     }&
3322     \def\babeltempc{}&
3323     \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&
3324     \ifin@ \else
3325       \bbl@xin@{:,\babeltempa,},{,\bbl@KVP@transforms,}&
3326     \fi
3327     \ifin@
3328       \bbl@foreach\bbl@KVP@transforms{%&
3329         \bbl@xin@{:,\babeltempa,},{,##1,}&
3330         \ifin@ & font:font:transform syntax
3331         \directlua{
3332           local t = {}
3333           for m in string.gmatch('##1'..'':, '(.-):') do
3334             table.insert(t, m)
3335           end
3336           table.remove(t)

```

```

3337         token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3338     }&%
3339     \fi}&%
3340     \in@{.0$}{#2$}&%
3341     \ifin@
3342         \directlua{&% (\attribute) syntax
3343             local str = string.match([[ \bbl@KVP@transforms]],
3344                 '%([^(%[-])%([^-])-\babeltempa')
3345             if str == nil then
3346                 token.set_macro('babeltempb', '')
3347             else
3348                 token.set_macro('babeltempb', ',attribute=' .. str)
3349             end
3350         }&%
3351         \toks@{#3}&%
3352         \bbl@exp{&%
3353             \\g@addto@macro\\bbl@release@transforms{&%
3354                 \relax &% Closes previous \bbl@transforms@aux
3355                 \\bbl@transforms@aux
3356                 \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3357                 {\language\the\toks@}}&%
3358         \else
3359             \g@addto@macro\bbl@release@transforms{, {#3}}&%
3360         \fi
3361     \fi}
3362 \endgroup

```

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

```

3363 \def\bbl@provide@lsys#1{%
3364     \bbl@ifunset{bbl@lname@#1}%
3365         {\bbl@load@info{#1}}%
3366     {%
3367         \bbl@csarg\let{lsys@#1}\@empty
3368         \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3369         \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3370         \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3371         \bbl@ifunset{bbl@lname@#1}{%
3372             {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3373         \ifcase\bbl@engine\or\or
3374             \bbl@ifunset{bbl@prehc@#1}{%
3375                 {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3376                 {%
3377                     {\ifx\bbl@xenohyph\undefined
3378                         \global\let\bbl@xenohyph\bbl@xenohyph@d
3379                         \ifx\AtBeginDocument\@notprerr
3380                             \expandafter\@secondoftwo % to execute right now
3381                         \fi
3382                         \AtBeginDocument{%
3383                             \bbl@patchfont{\bbl@xenohyph}%
3384                             \expandafter\select@language\expandafter{\language}%
3385                         \fi}}%
3386                 \fi
3387             \bbl@csarg\bbl@tglobal{lsys@#1}}
3388 \def\bbl@xenohyph@d{%
3389     \bbl@ifset{bbl@prehc@language}%
3390         {\ifnum\hyphenchar\font=\defaultthyphenchar
3391             \iffontchar\font\bbl@cl{prehc}\relax
3392             \hyphenchar\font\bbl@cl{prehc}\relax
3393         \else\iffontchar\font"200B
3394             \hyphenchar\font"200B
3395         \else
3396             \bbl@warning

```

```

3397         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3398         in the current font, and therefore the hyphen\\%
3399         will be printed. Try changing the fontspec's\\%
3400         'HyphenChar' to another value, but be aware\\%
3401         this setting is not safe (see the manual).\\%
3402         Reported}%
3403         \hyphenchar\font\defaultshyphenchar
3404         \fi\fi
3405         \fi}%
3406         {\hyphenchar\font\defaultshyphenchar}}
3407         % \fi}

```

```

3408 \def\bbl@load@info#1{%
3409   \def\BabelBeforeIni##1##2{%
3410     \begingroup
3411       \bbl@read@ini{##1}0%
3412       \endinput           % babel- .tex may contain onlypreamble's
3413       \endgroup}%        boxed, to avoid extra spaces:
3414   {\bbl@input@texini{#1}}}
```

```

3415 \def\bbl@setdigits#1#2#3#4#5{%
3416   \bbl@exp{%
3417     \def<\<language name digits>####1{%       ie, \langdigits
3418       \<bbl@digits@\language name>####1\\\@nil}%
3419     \let<\bbl@cntr@digits@\language name>\<\language name digits>%
3420     \def<\<language name counter>####1{%       ie, \langcounter
3421       \\\expandafter<\bbl@counter@\language name>%
3422       \\\csname c@####1\endcsname}%
3423     \def<\bbl@counter@\language name>####1{% ie, \bbl@counter@lang
3424       \\\expandafter<\bbl@digits@\language name>%
3425       \\\number####1\\\@nil}}}%
3426 \def\bbl@tempa##1##2##3##4##5{%
3427   \bbl@exp{%    Wow, quite a lot of hashes! :- (
3428     \def<\bbl@digits@\language name>#####1{%
3429       \\\ifx#####1\\\@nil                % ie, \bbl@digits@lang
3430       \\\else
3431         \\\ifx0#####1#1%
3432         \\\else\\\ifx1#####1#2%
3433         \\\else\\\ifx2#####1#3%
3434         \\\else\\\ifx3#####1#4%
3435         \\\else\\\ifx4#####1#5%
3436         \\\else\\\ifx5#####1##1%
3437         \\\else\\\ifx6#####1##2%
3438         \\\else\\\ifx7#####1##3%
3439         \\\else\\\ifx8#####1##4%
3440         \\\else\\\ifx9#####1##5%
3441         \\\else#####1%
3442         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3443         \\\expandafter<\bbl@digits@\language name>%
3444         \\\fi}}}%
3445 \bbl@tempa}

```

```
3446 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}  
3447   \ifx\##1%           % \\ before, in case #1 is multiletter  
3448     \bbl@exp{%  
3449       \def\\bbl@tempa####1{%
```

```

3450     \<ifcase>####1\space\the\toks@<\<else>\\\@ctrerr<\<fi>}}%
3451 \else
3452     \toks@<\expandafter{\the\toks@<\or #1}%
3453     \expandafter\bbbl@buildifcase
3454 \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).

```

3455 \newcommand\localecnumeral[2]{\bbbl@cs{cntnr@#1@<\language>name}{#2}}
3456 \def\bbbl@localecnumeral#1#2{\localecnumeral{#2}{#1}}
3457 \newcommand\localecounter[2]{%
3458     \expandafter\bbbl@localecnum
3459     \expandafter{\number\csname c@#2\endcsname}{#1}}
3460 \def\bbbl@alphnumeral#1#2{%
3461     \expandafter\bbbl@alphnumeral@i\number#2 76543210@@{#1}}
3462 \def\bbbl@alphnumeral@i#1#2#3#4#5#6#7#8\@#9{%
3463     \ifcase\@car#8\@nil\or % Currenty <10000, but prepared for bigger
3464         \bbbl@alphnumeral@ii{#9}000000#1\or
3465         \bbbl@alphnumeral@ii{#9}00000#1#2\or
3466         \bbbl@alphnumeral@ii{#9}0000#1#2#3\or
3467         \bbbl@alphnumeral@ii{#9}000#1#2#3#4\else
3468         \bbbl@alphnum@invalid{>9999}%
3469     \fi}
3470 \def\bbbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3471     \bbbl@ifunset{\bbbl@cntnr@#1.F.\number#5#6#7#8@<\language>name}%
3472     {\bbbl@cs{cntnr@#1.4@<\language>name}{#5}
3473     \bbbl@cs{cntnr@#1.3@<\language>name}{#6}
3474     \bbbl@cs{cntnr@#1.2@<\language>name}{#7}
3475     \bbbl@cs{cntnr@#1.1@<\language>name}{#8}
3476     \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3477         \bbbl@ifunset{\bbbl@cntnr@#1.S.321@<\language>name}{}}%
3478         {\bbbl@cs{cntnr@#1.S.321@<\language>name}{}}%
3479     \fi}%
3480     {\bbbl@cs{cntnr@#1.F.\number#5#6#7#8@<\language>name}{}}
3481 \def\bbbl@alphnum@invalid#1{%
3482     \bbbl@error{Alphabetic numeral too large (#1)}%
3483     {Currently this is the limit.}}

```

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

```

3484 \def\bbbl@localeinfo#1#2{%
3485     \bbbl@ifunset{\bbbl@info@#2}{#1}%
3486     {\bbbl@ifunset{\bbbl@csname bbl@info@#2\endcsname @<\language>name}{#1}%
3487     {\bbbl@cs{\csname bbl@info@#2\endcsname @<\language>name}{}}}
3488 \newcommand\localeinfo[1]{%
3489     \ifx*#1\empty % TODO. A bit hackish to make it expandable.
3490         \bbbl@afterelse\bbbl@localeinfo{}%
3491     \else
3492         \bbbl@localeinfo
3493         {\bbbl@error{I've found no info for the current locale.\\%
3494             The corresponding ini file has not been loaded\\%
3495             Perhaps it doesn't exist}%
3496         {See the manual for details.}}%
3497     {#1}%
3498 \fi}
3499 % \@namedef{\bbbl@info@name.locale}{\lcname}
3500 \@namedef{\bbbl@info@tag.ini}{\lini}
3501 \@namedef{\bbbl@info@name.english}{\elname}
3502 \@namedef{\bbbl@info@name.opentype}{\lname}
3503 \@namedef{\bbbl@info@tag.bcp47}{\tbcpl}
3504 \@namedef{\bbbl@info@language.tag.bcp47}{\lbcp}

```

```

3505 \@namedef{bbl@info@tag.opentype}{lotf}
3506 \@namedef{bbl@info@script.name}{esname}
3507 \@namedef{bbl@info@script.name.opentype}{sname}
3508 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3509 \@namedef{bbl@info@script.tag.opentype}{sotf}
3510 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3511 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3512 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3513 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3514 \@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.

```

3515 \providecommand\BCPdata{}
3516 \ifx\renewcommand\undefined\else % For plain. TODO. It's a quick fix
3517   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3518   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3519     \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3520     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3521     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}%
3522   \def\bbl@bcpdata@ii#1#2{%
3523     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3524     {\bbl@error{Unknown field '#1' in \string\BCPdata.\\%
3525       Perhaps you misspelled it.}%
3526     {See the manual for details.}}%
3527     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}}%
3528     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3529 \fi
3530 % Still somewhat hackish. WIP.
3531 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3532 \newcommand\BabelUppercaseMapping[3]{%
3533   \let\bbl@temp\language
3534   \edef\language{#1}%
3535   \DeclareUppercaseMapping[\BCPdata{casing}]{#2}{#3}%
3536   \let\language\bbl@temp}
3537 \newcommand\BabelLowercaseMapping[3]{%
3538   \let\bbl@temp\language
3539   \edef\language{#1}%
3540   \DeclareLowercaseMapping[\BCPdata{casing}]{#2}{#3}%
3541   \let\language\bbl@temp}

```

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

```

3542 <<More package options>> \=
3543 \DeclareOption{ensureinfo=off}{}
3544 <</More package options>>
3545 \let\bbl@ensureinfo@gobble
3546 \newcommand\BabelEnsureInfo{%
3547   \ifx\InputIfFileExists\undefined\else
3548     \def\bbl@ensureinfo##1{%
3549       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3550   \fi
3551   \bbl@foreach\bbl@loaded{{%
3552     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3553     \def\language{##1}%
3554     \bbl@ensureinfo{##1}}}
3555   \ifpackagewith{babel}{ensureinfo=off}{}%
3556   {\AtEndOfPackage{Test for plain.
3557     \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`.

```

3558 \newcommand\getlocaleproperty{%

```

```

3559 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3560 \def\bbl@getproperty@s#1#2#3{%
3561   \let#1\relax
3562   \def\bbl@elt##1##2##3{%
3563     \bbl@ifsamestring{##1/##2}{#3}%
3564     {\providecommand#1{##3}%
3565     \def\bbl@elt###1####2####3{}}}%
3566   {}}%
3567   \bbl@cs{inidata@#2}}}%
3568 \def\bbl@getproperty@x#1#2#3{%
3569   \bbl@getproperty@s{#1}{#2}{#3}%
3570   \ifx#1\relax
3571     \bbl@error
3572     {Unknown key for locale '#2':\%
3573     #3\}%
3574     \string#1 will be set to \relax}%
3575     {Perhaps you misspelled it.}%
3576   \fi}
3577 \let\bbl@ini@loaded\@empty
3578 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

5 Adjusting the Babel behavior

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

```

3579 \newcommand\babeladjust[1]{% TODO. Error handling.
3580   \bbl@forkv{#1}{%
3581     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3582     {\bbl@cs{ADJ@##1}{##2}}%
3583     {\bbl@cs{ADJ@##1@##2}}}
3584 %
3585 \def\bbl@adjust@lua#1#2{%
3586   \ifvmode
3587     \ifnum\currentgrouplevel=\z@
3588       \directlua{ Babel.#2 }%
3589       \expandafter\expandafter\expandafter\@gobble
3590     \fi
3591   \fi
3592   {\bbl@error % The error is gobbled if everything went ok.
3593     {Currently, #1 related features can be adjusted only\%
3594     in the main vertical list.}%
3595     {Maybe things change in the future, but this is what it is.}}}
3596 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3597   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3598 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3599   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3600 \@namedef{\bbl@ADJ@bidi.text@on}{%
3601   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3602 \@namedef{\bbl@ADJ@bidi.text@off}{%
3603   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3604 \@namedef{\bbl@ADJ@bidi.math@on}{%
3605   \let\bbl@noamsmath\@empty}
3606 \@namedef{\bbl@ADJ@bidi.math@off}{%
3607   \let\bbl@noamsmath\relax}
3608 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3609   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3610 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3611   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3612 %
3613 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3614   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3615 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3616   \bbl@adjust@lua{linebreak}{sea_enabled=false}}

```

```

3617 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3618   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3619 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3620   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3621 \@namedef{bbl@ADJ@justify.arabic@on}{%
3622   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3623 \@namedef{bbl@ADJ@justify.arabic@off}{%
3624   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3625 %
3626 \def\bbl@adjust@layout#1{%
3627   \ifvmode
3628     #1%
3629     \expandafter\@gobble
3630   \fi
3631   {\bbl@error   % The error is gobbled if everything went ok.
3632     {Currently, layout related features can be adjusted only\\%
3633       in vertical mode.}%
3634     {Maybe things change in the future, but this is what it is.}}}
3635 \@namedef{bbl@ADJ@layout.tabular@on}{%
3636   \ifnum\bbl@tabular@mode=\tw@
3637     \bbl@adjust@layout{\let\@tabular\bbl@NL@\@tabular}%
3638   \else
3639     \chardef\bbl@tabular@mode\@ne
3640   \fi}
3641 \@namedef{bbl@ADJ@layout.tabular@off}{%
3642   \ifnum\bbl@tabular@mode=\tw@
3643     \bbl@adjust@layout{\let\@tabular\bbl@OL@\@tabular}%
3644   \else
3645     \chardef\bbl@tabular@mode\z@
3646   \fi}
3647 \@namedef{bbl@ADJ@layout.lists@on}{%
3648   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3649 \@namedef{bbl@ADJ@layout.lists@off}{%
3650   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3651 %
3652 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3653   \bbl@bcpallowedtrue}
3654 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3655   \bbl@bcpallowedfalse}
3656 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
3657   \def\bbl@bcp@prefix{#1}}
3658 \def\bbl@bcp@prefix{bcp47-}
3659 \@namedef{bbl@ADJ@autoload.options#1}{%
3660   \def\bbl@autoload@options{#1}}
3661 \let\bbl@autoload@bcptoptions\@empty
3662 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
3663   \def\bbl@autoload@bcptoptions{#1}}
3664 \newif\ifbbl@bcptoname
3665 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3666   \bbl@bcptonamettrue}
3667 \BabelEnsureInfo}
3668 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3669   \bbl@bcptonamefalse}
3670 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3671   \directlua{ Babel.ignore_pre_char = function(node)
3672     return (node.lang == \the\csname l@nohyphenation\endcsname)
3673   end }}
3674 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3675   \directlua{ Babel.ignore_pre_char = function(node)
3676     return false
3677   end }}
3678 \@namedef{bbl@ADJ@select.write@shift}{%
3679   \let\bbl@restorelastskip\relax

```

```

3680 \def\bbl@savelastskip{%
3681   \let\bbl@restorelastskip\relax
3682   \ifvmode
3683     \ifdim\lastskip=\z@
3684       \let\bbl@restorelastskip\nobreak
3685     \else
3686       \bbl@exp{%
3687         \def\\bbl@restorelastskip{%
3688           \skip@=\the\lastskip
3689           \\nobreak \vskip-\skip@ \vskip\skip@}}%
3690     \fi
3691   \fi}}
3692 \@namedef{bbl@ADJ@select.write@keep}{%
3693   \let\bbl@restorelastskip\relax
3694   \let\bbl@savelastskip\relax}
3695 \@namedef{bbl@ADJ@select.write@omit}{%
3696   \AddBabelHook{babel-select}{beforestart}{%
3697     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3698 \let\bbl@restorelastskip\relax
3699 \def\bbl@savelastskip##1\bbl@restorelastskip{}
3700 \@namedef{bbl@ADJ@select.encoding@off}{%
3701   \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.

```

3702 <<More package options>> ≡
3703 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3704 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3705 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3706 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3707 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3708 <</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.

```

3709 \bbl@trace{Cross referencing macros}
3710 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3711   \def\@newl@bel#1#2#3{%
3712     {\@safe@activestrue
3713       \bbl@ifunset{#1@#2}%
3714       \relax
3715       {\gdef\@multiplelabels{%
3716         \@latex@warning@no@line{There were multiply-defined labels}}%
3717         \@latex@warning@no@line{Label `#2' multiply defined}}%
3718       \global\@namedef{#1@#2}{#3}}}

```

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

```

3719 \CheckCommand*\@testdef[3]{%
3720   \def\reserved@a{#3}%
3721   \expandafter\ifx\c@name#1@#2\endcsname\reserved@a
3722   \else

```



```

3723     \@tempswattrue
3724     \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.

```

3725 \def\@testdef#1#2#3{% TODO. With @samestring?
3726     \@safe@activetrue
3727     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3728     \def\bbl@tempb{#3}%
3729     \@safe@activesfalse
3730     \ifx\bbl@tempa\relax
3731     \else
3732         \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3733     \fi
3734     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3735     \ifx\bbl@tempa\bbl@tempb
3736     \else
3737         \@tempswattrue
3738     \fi}
3739 \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.

```

3740 \bbl@xin@{R}\bbl@opt@safe
3741 \ifin@
3742     \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3743     \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3744     {\expandafter\strip@prefix\meaning\ref}%
3745 \ifin@
3746     \bbl@redefine\@kernel@ref#1{%
3747         \@safe@activetrue\org@@kernel@ref{#1}\@safe@activesfalse}
3748     \bbl@redefine\@kernel@pageref#1{%
3749         \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3750     \bbl@redefine\@kernel@sref#1{%
3751         \@safe@activetrue\org@@kernel@sref{#1}\@safe@activesfalse}
3752     \bbl@redefine\@kernel@spageref#1{%
3753         \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3754 \else
3755     \bbl@redefinero bust\ref#1{%
3756         \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3757     \bbl@redefinero bust\pageref#1{%
3758         \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3759 \fi
3760 \else
3761     \let\org@ref\ref
3762     \let\org@pageref\pageref
3763 \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.

```

3764 \bbl@xin@{B}\bbl@opt@safe
3765 \ifin@
3766     \bbl@redefine\@citex[#1]#2{%
3767         \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3768         \org@@citex[#1]{\@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.

```
3769 \AtBeginDocument{%
3770   \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.)

```
3771   \def\@citex[#1][#2]#3{%
3772     \@safe@activestruedef\@tempa{#3}\@safe@activestruedefalse
3773     \org@citex[#1][#2]{\@tempa}}%
3774   }{}}
```

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

```
3775 \AtBeginDocument{%
3776   \ifpackageloaded{cite}{%
3777     \def\@citex[#1]#2{%
3778       \@safe@activestruedef\org@citex[#1][#2]\@safe@activestruedefalse}%
3779     }{}}
```

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

```
3780 \bbl@redefine\nocite#1{%
3781   \@safe@activestruedef\org@nocite{#1}\@safe@activestruedefalse}
```

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

```
3782 \bbl@redefine\bibcite{%
3783   \bbl@cite@choice
3784   \bibcite}
```

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

```
3785 \def\bbl@bibcite#1#2{%
3786   \org@bibcite{#1}{\@safe@activestruedefalse#2}}
```

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

```
3787 \def\bbl@cite@choice{%
3788   \global\let\bibcite\bbl@bibcite
3789   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3790   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3791   \global\let\bbl@cite@choice\relax}
```

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

```
3792 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3793 \bbl@redefine\@bibitem#1{%
3794   \@safe@activestruedef\org@bibitem{#1}\@safe@activestruedefalse}
3795 \else
3796   \let\org@nocite\nocite
3797   \let\org@citex\@citex
3798   \let\org@bibcite\bibcite
3799   \let\org@bibitem\@bibitem
3800 \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.

```
3801 \bbl@trace{Marks}
3802 \IfBabelLayout{sectioning}
3803   {\ifx\bbl@opt@headfoot\@nnil
3804     \g@addto@macro\@resetactivechars{%
3805       \set@typeset@protect
3806       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3807       \let\protect\noexpand
3808       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3809         \edef\thepage{%
3810           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3811       \fi}%
3812   \fi}
3813 {\ifbbl@single\else
3814   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3815   \markright#1{%
3816     \bbl@ifblank{#1}%
3817     {\org@markright{}}%
3818     {\toks@{#1}%
3819       \bbl@exp{%
3820         \\org@markright{\\protect\\foreignlanguage{\language}\language}%
3821         {\\protect\\bbl@restore@actives\the\toks@}}}%
3822   }
```

`\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.)

```
3822   \ifx\@mkboth\markboth
3823     \def\bbl@tempc{\let\@mkboth\markboth}%
3824   \else
3825     \def\bbl@tempc{%
3826       \fi
3827       \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3828       \markboth#1#2{%
3829         \protected@edef\bbl@tempb##1{%
3830           \protect\foreignlanguage
3831             {\language}\protect\bbl@restore@actives##1}%
3832         \bbl@ifblank{#1}%
3833         {\toks@{}}%
3834         {\toks@\expandafter{\bbl@tempb{#1}}}%
3835         \bbl@ifblank{#2}%
3836         {\@temptokena{}}%
3837         {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3838         \bbl@exp{\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3839         \bbl@tempc
3840       \fi} % end ifbbl@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.

```

3841 \bbl@trace{Preventing clashes with other packages}
3842 \ifx\org@ref\undefined\else
3843   \bbl@xin@{R}\bbl@opt@safe
3844   \ifin@
3845     \AtBeginDocument{%
3846       \ifpackageloaded{ifthen}{%
3847         \bbl@redefine@long\ifthenelse#1#2#3{%
3848           \let\bbl@temp@pref\pageref
3849           \let\pageref\org@pageref
3850           \let\bbl@temp@ref\ref
3851           \let\ref\org@ref
3852           \@safe@activestrue
3853           \org@ifthenelse{#1}%
3854             {\let\pageref\bbl@temp@pref
3855              \let\ref\bbl@temp@ref
3856              \@safe@activesfalse
3857              #2}%
3858             {\let\pageref\bbl@temp@pref
3859              \let\ref\bbl@temp@ref
3860              \@safe@activesfalse
3861              #3}%
3862           }%
3863         }{}%
3864       }
3865 \fi

```

5.3.2 varioref

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

```

3866 \AtBeginDocument{%
3867   \@ifpackageloaded{varioref}{%
3868     \bbl@redefine\@vppageref#1[#2]#3{%
3869       \@safe@activestrue
3870       \org@@@vppageref{#1}[#2][#3}%
3871       \@safe@activesfalse}%
3872   \bbl@redefine\vrefpagemum#1#2{%
3873     \@safe@activestrue
3874     \org@vrefpagemum{#1}[#2}%
3875     \@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.

```

3876   \expandafter\def\csname Ref \endcsname#1{%
3877     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3878   }{}%

```

```

3879     }
3880 \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.

```

3881 \AtEndOfPackage{%
3882   \AtBeginDocument{%
3883     \ifpackageloaded{hhline}%
3884       {\expandafter\ifx\csname normal@char\string\endcsname\relax
3885         \else
3886           \makeatletter
3887           \def\@currname{hhline}\input{hhline.sty}\makeatother
3888         \fi}%
3889     {}%

```

`\substitutefontfamily` *Deprecated.* Use the tools provides by \LaTeX . 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.

```

3890 \def\substitutefontfamily#1#2#3{%
3891   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3892   \immediate\write15{%
3893     \string\ProvidesFile{#1#2.fd}%
3894     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3895     \space generated font description file]^{}
3896     \string\DeclareFontFamily{#1}{#2}{}}^{}
3897     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}}^{}
3898     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}}^{}
3899     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}}^{}
3900     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}}^{}
3901     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}}^{}
3902     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}}^{}
3903     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}}^{}
3904     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}^{}
3905   }%
3906   \closeout15
3907 }
3908 \@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`

```

3909 \bbl@trace{Encoding and fonts}
3910 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3911 \newcommand\BabelNonText{TS1,T3,TS3}
3912 \let\org@TeX\TeX
3913 \let\org@LaTeX\LaTeX
3914 \let\ensureascii\@firstofone
3915 \AtBeginDocument{%
3916   \def\@elt#1{,#1,}%
3917   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3918   \let\@elt\relax

```

```

3919 \let\bbl@tempb\@empty
3920 \def\bbl@tempc{OT1}%
3921 \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3922   \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3923 \bbl@foreach\bbl@tempa{%
3924   \bbl@xin@{#1}{\BabelNonASCII}%
3925   \ifin@
3926     \def\bbl@tempb{#1}% Store last non-ascii
3927   \else\bbl@xin@{#1}{\BabelNonText}% Pass
3928     \ifin@else
3929       \def\bbl@tempc{#1}% Store last ascii
3930     \fi
3931   \fi}%
3932 \ifx\bbl@tempb\@empty\else
3933   \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3934   \ifin@else
3935     \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3936   \fi
3937   \edef\ensureascii#1{%
3938     {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3939   \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3940   \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3941 \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.

```

3942 \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.

```

3943 \AtBeginDocument{%
3944   \ifpackageloaded{fontspec}%
3945     {\xdef\latinencoding{%
3946       \ifx\UTFfencname\@undefined
3947         EU\ifcase\bbl@engine\or2\or1\fi
3948       \else
3949         \UTFfencname
3950       \fi}}%
3951   {\gdef\latinencoding{OT1}%
3952     \ifx\cf@encoding\bbl@t@one
3953       \xdef\latinencoding{\bbl@t@one}%
3954     \else
3955       \def\@elt#1{, #1,}%
3956       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3957       \let\@elt\relax
3958       \bbl@xin@{,T1,}\bbl@tempa
3959       \ifin@
3960         \xdef\latinencoding{\bbl@t@one}%
3961       \fi
3962     \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.

```

3963 \DeclareRobustCommand{\latintext}{%
3964   \fontencoding{\latinencoding}\selectfont
3965   \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.

```

3966 \ifx\@undefined\DeclareTextFontCommand
3967   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3968 \else
3969   \DeclareTextFontCommand{\textlatin}{\latintext}
3970 \fi

```

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

```

3971 \def\bbl@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 T_EX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `LuaTEX-jā` shows, vertical typesetting is possible, too.

```

3972 \bbl@trace{Loading basic (internal) bidi support}
3973 \ifodd\bbl@engine
3974 \else % TODO. Move to txtbabel
3975   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any lua bidi=
3976     \bbl@error
3977     {The bidi method 'basic' is available only in\\%
3978     luatex. I'll continue with 'bidi=default', so\\%
3979     expect wrong results}%
3980     {See the manual for further details.}%
3981     \let\bbl@beforeforeign\leavevmode
3982     \AtEndOfPackage{%
3983       \EnableBabelHook{babel-bidi}%
3984       \bbl@xebidipar}
3985   \fi\fi
3986   \def\bbl@loadxebidi#1{%
3987     \ifx\RTLfootnotetext\@undefined
3988       \AtEndOfPackage{%
3989         \EnableBabelHook{babel-bidi}%
3990         \bbl@loadfontspec % bidi needs fontspec
3991         \usepackage#1{bidi}}%
3992     \fi}
3993   \ifnum\bbl@bidimode>200 % Any xe bidi=
3994     \ifcase\expandafter\gobbletwo\the\bbl@bidimode\or
3995       \bbl@tentative{bidi=bidi}
3996       \bbl@loadxebidi{}
3997     \or
3998       \bbl@loadxebidi{[rldocument]}
3999     \or

```

```

4000     \bbl@loadxebidi{}
4001   \fi
4002 \fi
4003 \fi
4004 % TODO? Separate:
4005 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
4006   \let\bbl@beforeforeign\leavevmode
4007   \ifodd\bbl@engine
4008     \newattribute\bbl@attr@dir
4009     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4010     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4011   \fi
4012 \AtEndOfPackage{%
4013   \EnableBabelHook{babel-bidi}%
4014   \ifodd\bbl@engine\else
4015     \bbl@xebidipar
4016   \fi}
4017 \fi

```

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

```

4018 \bbl@trace{Macros to switch the text direction}
4019 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4020 \def\bbl@rscripts{% TODO. Base on codes ??
4021   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4022   Old Hungarian,Lydian,Mandaean,Manichaeen,%
4023   Meroitic Cursive,Meroitic,Old North Arabian,%
4024   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4025   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4026   Old South Arabian,}%
4027 \def\bbl@provide@dirs#1{%
4028   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4029   \ifin@
4030     \global\bbl@csarg\chardef{wdir@#1}\@ne
4031     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4032     \ifin@
4033       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4034     \fi
4035   \else
4036     \global\bbl@csarg\chardef{wdir@#1}\z@
4037   \fi
4038   \ifodd\bbl@engine
4039     \bbl@csarg\ifcase{wdir@#1}%
4040       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4041     \or
4042       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4043     \or
4044       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4045     \fi
4046   \fi}
4047 \def\bbl@switchdir{%
4048   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4049   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4050   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
4051 \def\bbl@setdirs#1{% TODO - math
4052   \ifcase\bbl@select@type % TODO - strictly, not the right test
4053     \bbl@bodydir{#1}%
4054     \bbl@paddir{#1}% <- Must precede \bbl@texdir
4055   \fi
4056   \bbl@texdir{#1}}
4057 % TODO. Only if \bbl@bidimode > 0?:
4058 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4059 \DisableBabelHook{babel-bidi}

```


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

```

4060 \ifodd\bb@engine % luatex=1
4061 \else % pdftex=0, xetex=2
4062   \newcount\bb@dirlevel
4063   \chardef\bb@thetextdir\z@
4064   \chardef\bb@thepardir\z@
4065   \def\bb@textdir#1{%
4066     \ifcase#1\relax
4067       \chardef\bb@thetextdir\z@
4068       \bb@textdir@i\beginL\endL
4069     \else
4070       \chardef\bb@thetextdir\@ne
4071       \bb@textdir@i\beginR\endR
4072     \fi}
4073   \def\bb@textdir@i#1#2{%
4074     \ifhmode
4075       \ifnum\currentgrouplevel>\z@
4076         \ifnum\currentgrouplevel=\bb@dirlevel
4077           \bb@error{Multiple bidi settings inside a group}%
4078           {I'll insert a new group, but expect wrong results.}%
4079           \bgroup\aftergroup#2\aftergroup\egroup
4080         \else
4081           \ifcase\currentgrouptype\or % 0 bottom
4082             \aftergroup#2% 1 simple {}
4083           \or
4084             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4085           \or
4086             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4087           \or\or\or % vbox vtop align
4088           \or
4089             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4090           \or\or\or\or\or\or % output math disc insert vcent mathchoice
4091           \or
4092             \aftergroup#2% 14 \begingroup
4093           \else
4094             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4095           \fi
4096         \fi
4097         \bb@dirlevel\currentgrouplevel
4098       \fi
4099       #1%
4100     \fi}
4101   \def\bb@pardir#1{\chardef\bb@thepardir#1\relax}
4102   \let\bb@bodydir\@gobble
4103   \let\bb@pagedir\@gobble
4104   \def\bb@dirparastext{\chardef\bb@thepardir\bb@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).

```

4105 \def\bb@xebidipar{%
4106   \let\bb@xebidipar\relax
4107   \TeXeTstate\@ne
4108   \def\bb@xeverypar{%
4109     \ifcase\bb@thepardir
4110       \ifcase\bb@thetextdir\else\beginR\fi
4111     \else
4112       {\setbox\z@\lastbox\beginR\box\z@}%
4113     \fi}%
4114   \let\bb@severypar\everypar
4115   \newtoks\everypar
4116   \everypar=\bb@severypar
4117   \bb@severypar{\bb@xeverypar\the\everypar}}

```

```

4118 \ifnum\bbl@bidimode>200 % Any xe bidi=
4119 \let\bbl@textdir@i\@gobbletwo
4120 \let\bbl@xebidipar\@empty
4121 \AddBabelHook{bidi}{foreign}{%
4122 \def\bbl@tempa{\def\BabelText####1}%
4123 \ifcase\bbl@thetextdir
4124 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4125 \else
4126 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4127 \fi}
4128 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4129 \fi
4130 \fi

```

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

```

4131 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4132 \AtBeginDocument{%
4133 \ifx\pdfstringdefDisableCommands\@undefined\else
4134 \ifx\pdfstringdefDisableCommands\relax\else
4135 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4136 \fi
4137 \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`.

```

4138 \bbl@trace{Local Language Configuration}
4139 \ifx\loadlocalcfg\@undefined
4140 \@ifpackagewith{babel}{noconfigs}%
4141 {\let\loadlocalcfg\@gobble}%
4142 {\def\loadlocalcfg#1{%
4143 \InputIfFileExists{#1.cfg}%
4144 {\typeout{*****^J%
4145 * Local config file #1.cfg used^^J%
4146 *}}}%
4147 \@empty}}
4148 \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).

```

4149 \bbl@trace{Language options}
4150 \let\bbl@afterlang\relax
4151 \let\BabelModifiers\relax
4152 \let\bbl@loaded\@empty
4153 \def\bbl@load@language#1{%
4154 \InputIfFileExists{#1.ldf}%
4155 {\edef\bbl@loaded{\CurrentOption
4156 \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4157 \expandafter\let\expandafter\bbl@afterlang
4158 \csname\CurrentOption.ldf-h@k\endcsname
4159 \expandafter\let\expandafter\BabelModifiers
4160 \csname\bbl@mod@\CurrentOption\endcsname
4161 \bbl@exp{\@AtBeginDocument{%
4162 \\\bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4163 {\bbl@error{%

```

```

4164     Unknown option '\CurrentOption'. Either you misspelled it\\%
4165     or the language definition file \CurrentOption.ldf was not found}{%
4166     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4167     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4168     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

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.

```

4169 \def\bbl@try@load@lang#1#2#3{%
4170   \IfFileExists{\CurrentOption.ldf}%
4171   {\bbl@load@language{\CurrentOption}}%
4172   {#1\bbl@load@language{#2}#3}}
4173 %
4174 \DeclareOption{hebrew}{%
4175   \input{rlbabel.def}%
4176   \bbl@load@language{hebrew}}
4177 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4178 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4179 \DeclareOption{northensami}{\bbl@try@load@lang{}{samin}{}}
4180 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4181 \DeclareOption{polutonikogreek}{%
4182   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}
4183   \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4184   \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4185   \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4186   \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.

```

4187 \ifx\bbl@opt@config@nnil
4188   \ifpackagewith{babel}{noconfigs}{}%
4189   {\InputIfFileExists{bblopts.cfg}%
4190     {\typeout{*****^J%
4191               * Local config file bblopts.cfg used^^J%
4192               *}}}%
4193   {}}%
4194 \else
4195   \InputIfFileExists{\bbl@opt@config.cfg}%
4196   {\typeout{*****^J%
4197             * Local config file \bbl@opt@config.cfg used^^J%
4198             *}}%
4199   {\bbl@error{%
4200     Local config file '\bbl@opt@config.cfg' not found}{%
4201     Perhaps you misspelled it.}}%
4202 \fi
```

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

```

4203 \ifx\bbl@opt@main\@nnil
4204   \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4205     \let\bbl@tempb\@empty
4206     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4207     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4208     \bbl@foreach\bbl@tempb{%   \bbl@tempb is a reversed list
4209       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4210         \ifodd\bbl@iniflag % = *=
```

```

4211      \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4212      \else % n +=
4213      \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4214      \fi
4215    \fi}%
4216  \fi
4217 \else
4218  \bbl@info{Main language set with 'main='. Except if you have\\%
4219            problems, prefer the default mechanism for setting\\%
4220            the main language, ie, as the last declared.\\%
4221            Reported}
4222 \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).

```

4223 \ifx\bbl@opt@main\@nnil\else
4224   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4225   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4226 \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 correspondin file exists.

```

4227 \bbl@foreach\bbl@language@opts{%
4228   \def\bbl@tempa{#1}%
4229   \ifx\bbl@tempa\bbl@opt@main\else
4230     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4231       \bbl@ifunset{ds@#1}%
4232       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4233       {}%
4234     \else % + * (other = ini)
4235       \DeclareOption{#1}{%
4236         \bbl@ldfinit
4237         \babelprovide[import]{#1}%
4238         \bbl@afterldf{}}}%
4239     \fi
4240   \fi}
4241 \bbl@foreach\@classoptionslist{%
4242   \def\bbl@tempa{#1}%
4243   \ifx\bbl@tempa\bbl@opt@main\else
4244     \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4245       \bbl@ifunset{ds@#1}%
4246       {\IfFileExists{#1.ldf}%
4247        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4248        {}}}%
4249     {}%
4250   \else % + * (other = ini)
4251     \IfFileExists{babel-#1.tex}%
4252     {\DeclareOption{#1}{%
4253       \bbl@ldfinit
4254       \babelprovide[import]{#1}%
4255       \bbl@afterldf{}}}%
4256     {}%
4257   \fi
4258 \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 processes before):

```

4259 \def\AfterBabelLanguage#1{%
4260   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4261   \DeclareOption*{}
4262 \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.

```

4263 \bbl@trace{Option 'main'}
4264 \ifx\bbl@opt@main\@nnil
4265   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4266   \let\bbl@tempc\@empty
4267   \edef\bbl@templ{\,\bbl@loaded,}
4268   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4269   \bbl@for\bbl@tempb\bbl@tempa{%
4270     \edef\bbl@tempd{\,\bbl@tempb,}%
4271     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4272     \bbl@xin{\bbl@tempd}{\bbl@templ}%
4273     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4274   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4275   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4276   \ifx\bbl@tempb\bbl@tempc\else
4277     \bbl@warning{%
4278       Last declared language option is '\bbl@tempc',\%
4279       but the last processed one was '\bbl@tempb'.\%
4280       The main language can't be set as both a global\%
4281       and a package option. Use 'main=\bbl@tempc' as\%
4282       option. Reported}
4283   \fi
4284 \else
4285   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4286     \bbl@ldfinit
4287     \let\CurrentOption\bbl@opt@main
4288     \bbl@exp{% \bbl@opt@provide = empty if *}
4289     \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}%
4290     \bbl@afterldf{}
4291     \DeclareOption{\bbl@opt@main}{}
4292   \else % case 0,2 (main is ldf)
4293     \ifx\bbl@loadmain\relax
4294       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4295     \else
4296       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4297     \fi
4298     \ExecuteOptions{\bbl@opt@main}
4299     \@namedef{ds@\bbl@opt@main}{}%
4300   \fi
4301   \DeclareOption*{}
4302   \ProcessOptions*
4303 \fi
4304 \bbl@exp{%
4305   \\\AtBeginDocument{\bbl@usehooks@lang{/}{\begindocument}{}}}%
4306 \def\AfterBabelLanguage{%
4307   \bbl@error
4308     {Too late for \string\AfterBabelLanguage}%
4309     {Languages have been loaded, so I can do nothing}}

```

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.

```

4310 \ifx\bbl@main@language\@undefined
4311   \bbl@info{%
4312     You haven't specified a language as a class or package\%
4313     option. I'll load 'nil'. Reported}
4314   \bbl@load@language{nil}
4315 \fi
4316 \</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 TeX users might want to use some of the features of the babel system too, care has to be taken that plain TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain TeX and L^AT_EX, some of it is for the L^AT_EX 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

```
4317 <*kernel>
4318 \let\bbl@onlyswitch\@empty
4319 \input babel.def
4320 \let\bbl@onlyswitch\@undefined
4321 </kernel>
4322 <*patterns>
```

7 Loading hyphenation patterns

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

```
4323 <<Make sure ProvidesFile is defined>>
4324 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4325 \xdef\bbl@format{\jobname}
4326 \def\bbl@version{<<version>>}
4327 \def\bbl@date{<<date>>}
4328 \ifx\AtBeginDocument\@undefined
4329   \def\@empty{}
4330 \fi
4331 <<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.

```
4332 \def\process@line#1#2 #3 #4 {%
4333   \ifx=#1%
4334     \process@synonym{#2}%
4335   \else
4336     \process@language{#1#2}{#3}{#4}%
4337   \fi
4338   \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.

```
4339 \toks@{}
4340 \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.

```
4341 \def\process@synonym#1{%
4342   \ifnum\last@language=\m@ne
4343     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4344   \else
4345     \expandafter\chardef\csname l@#1\endcsname\last@language
```

```

4346 \wlog{\string\l@#1=\string\language\the\last@language}%
4347 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4348 \csname\language\hyphenmins\endcsname
4349 \let\bbl@elt\relax
4350 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4351 \fi}

```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language.

The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

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

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

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

`\bbl@elt{\langle language-name \rangle}{\langle number \rangle}{\langle patterns-file \rangle}{\langle exceptions-file \rangle}`. 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.

```

4352 \def\process@language#1#2#3{%
4353 \expandafter\addlanguage\csname l@#1\endcsname
4354 \expandafter\language\csname l@#1\endcsname
4355 \edef\language{#1}%
4356 \bbl@hook@everylanguage{#1}%
4357 % > luatex
4358 \bbl@get@enc#1::\@@@
4359 \begingroup
4360 \lefthyphenmin\m@ne
4361 \bbl@hook@loadpatterns{#2}%
4362 % > luatex
4363 \ifnum\lefthyphenmin=\m@ne
4364 \else
4365 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4366 \the\lefthyphenmin\the\righthyphenmin}%
4367 \fi
4368 \endgroup
4369 \def\bbl@tempa{#3}%
4370 \ifx\bbl@tempa\empty\else
4371 \bbl@hook@loadexceptions{#3}%
4372 % > luatex
4373 \fi
4374 \let\bbl@elt\relax
4375 \edef\bbl@languages{%
4376 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4377 \ifnum\the\language=\z@
4378 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4379 \set@hyphenmins\tw@\thr@@\relax
4380 \else

```

```

4381 \expandafter\expandafter\expandafter\set@hyphenmins
4382 \csname #1hyphenmins\endcsname
4383 \fi
4384 \the\toks@
4385 \toks@{}%
4386 \fi}

```

\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it in
\bbl@hyph@enc \bbl@hyph@enc. It uses delimited arguments to achieve this.

```

4387 \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.

```

4388 \def\bbl@hook@everylanguage#1{}
4389 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4390 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4391 \def\bbl@hook@loadkernel#1{%
4392 \def\addlanguage{\csname newlanguage\endcsname}%
4393 \def\adddialect##1##2{%
4394 \global\chardef##1##2\relax
4395 \wlog{\string##1 = a dialect from \string\language##2}}%
4396 \def\iflanguage##1{%
4397 \expandafter\ifx\csname l@##1\endcsname\relax
4398 \@nolanerr{##1}%
4399 \else
4400 \ifnum\csname l@##1\endcsname=\language
4401 \expandafter\expandafter\expandafter\@firstoftwo
4402 \else
4403 \expandafter\expandafter\expandafter\@secondoftwo
4404 \fi
4405 \fi}%
4406 \def\providehyphenmins##1##2{%
4407 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4408 \@namedef{##1hyphenmins}{##2}%
4409 \fi}%
4410 \def\set@hyphenmins##1##2{%
4411 \lefthyphenmin##1\relax
4412 \righthyphenmin##2\relax}%
4413 \def\selectlanguage{%
4414 \errhelp{Selecting a language requires a package supporting it}%
4415 \errmessage{Not loaded}}%
4416 \let\foreignlanguage\selectlanguage
4417 \let\otherlanguage\selectlanguage
4418 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4419 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4420 \def\setlocale{%
4421 \errhelp{Find an armchair, sit down and wait}%
4422 \errmessage{Not yet available}}%
4423 \let\uselocale\setlocale
4424 \let\locale\setlocale
4425 \let\selectlocale\setlocale
4426 \let\localename\setlocale
4427 \let\textlocale\setlocale
4428 \let\textlanguage\setlocale
4429 \let\languagetext\setlocale}
4430 \begingroup
4431 \def\AddBabelHook#1#2{%
4432 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4433 \def\next{\toks1}%
4434 \else
4435 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4436 \fi

```



```

4437 \next}
4438 \ifx\directlua\undefined
4439 \ifx\XeTeXinputencoding\undefined\else
4440 \input xebabel.def
4441 \fi
4442 \else
4443 \input luababel.def
4444 \fi
4445 \openin1 = babel-\bbl@format.cfg
4446 \ifeof1
4447 \else
4448 \input babel-\bbl@format.cfg\relax
4449 \fi
4450 \closein1
4451 \endgroup
4452 \bbl@hook@loadkernel{switch.def}

```

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

```

4453 \openin1 = language.dat

```

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

```

4454 \def\language{english}%
4455 \ifeof1
4456 \message{I couldn't find the file language.dat,\space
4457          I will try the file hyphen.tex}
4458 \input hyphen.tex\relax
4459 \chardef\l@english\z@
4460 \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`.

```

4461 \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.

```

4462 \loop
4463 \endlinechar\m@ne
4464 \read1 to \bbl@line
4465 \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.

```

4466 \if T\ifeof1\fi T\relax
4467 \ifx\bbl@line\empty\else
4468 \edef\bbl@line{\bbl@line\space\space\space}%
4469 \expandafter\process@line\bbl@line\relax
4470 \fi
4471 \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.

```

4472 \begingroup
4473 \def\bbl@elt#1#2#3#4{%
4474 \global\language=#2\relax
4475 \gdef\language{#1}%
4476 \def\bbl@elt##1##2##3##4{}}%
4477 \bbl@languages
4478 \endgroup
4479 \fi
4480 \closein1

```

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

```
4481 \if/\the\toks@/\else
4482   \errhelp{language.dat loads no language, only synonyms}
4483   \errmessage{Orphan language synonym}
4484 \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.

```
4485 \let\bbl@line\undefined
4486 \let\process@line\undefined
4487 \let\process@synonym\undefined
4488 \let\process@language\undefined
4489 \let\bbl@get@enc\undefined
4490 \let\bbl@hyph@enc\undefined
4491 \let\bbl@tempa\undefined
4492 \let\bbl@hook@loadkernel\undefined
4493 \let\bbl@hook@everylanguage\undefined
4494 \let\bbl@hook@loadpatterns\undefined
4495 \let\bbl@hook@loadexceptions\undefined
4496 \patterns)
```

Here the code for `iniTeX` 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].

```
4497 <<*More package options>> ≡
4498 \chardef\bbl@bidimode\z@
4499 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4500 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4501 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4502 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4503 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4504 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4505 <</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.

```
4506 <<*Font selection>> ≡
4507 \bbl@trace{Font handling with fontspec}
4508 \ifx\ExplSyntaxOn\undefined\else
4509   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4510     \in@{,#1,}{,no-script,language-not-exist,}%
4511     \ifin@ \else \bbl@tempfs@nx{#1}{#2}\fi}
4512   \def\bbl@fs@warn@nxx#1#2#3{%
4513     \in@{,#1,}{,no-script,language-not-exist,}%
4514     \ifin@ \else \bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4515   \def\bbl@loadfontspec{%
4516     \let\bbl@loadfontspec\relax
4517     \ifx\fontspec\undefined
4518       \usepackage{fontspec}%
4519     \fi}%
4520 \fi
4521 \@onlypreamble\babelfont
4522 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4523   \bbl@foreach{#1}{%
```

```

4524 \expandafter\ifx\csname date##1\endcsname\relax
4525 \IfFileExists{babel-##1.tex}%
4526 {\babelprovide{##1}}%
4527 {}%
4528 \fi}%
4529 \edef\bbl@tempa{#1}%
4530 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4531 \bbl@loadfontspec
4532 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4533 \bbl@bblfont}
4534 \newcommand\bbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4535 \bbl@ifunset{\bbl@tempb family}%
4536 {\bbl@providefam{\bbl@tempb}}%
4537 {}%
4538 % For the default font, just in case:
4539 \bbl@ifunset{\bbl@lsys@\language name}{\bbl@provide@lsys{\language name}}{}%
4540 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4541 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4542 \bbl@exp{%
4543 \let\bbl@bbl@tempb dflt@\language name>\<\bbl@bbl@tempb dflt@>%
4544 \bbl@font@set{\<\bbl@bbl@tempb dflt@\language name>%
4545 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4546 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4547 \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:

```

4548 \def\bbl@providefam#1{%
4549 \bbl@exp{%
4550 \\\newcommand\<#1default>{}% Just define it
4551 \\\bbl@add@list\\bbl@font@fams{#1}%
4552 \\\DeclareRobustCommand\<#1family>{%
4553 \\\not@math@alphabet\<#1family>\relax
4554 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4555 \\\fontfamily\<#1default>%
4556 \<ifx>\\\UseHooks\\undefined\<else>\\\UseHook{#1family}\<fi>%
4557 \\\selectfont}%
4558 \\\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.

```

4559 \def\bbl@nostdfont#1{%
4560 \bbl@ifunset{\bbl@WFF@\fontfamily}%
4561 {\bbl@csarg\gdef{WFF@\fontfamily}}% Flag, to avoid dupl warns
4562 \bbl@infowarn{The current font is not a babel standard family:\\%
4563 #1%
4564 \fontname\font\\%
4565 There is nothing intrinsically wrong with this warning, and\\%
4566 you can ignore it altogether if you do not need these\\%
4567 families. But if they are used in the document, you should be\\%
4568 aware 'babel' will not set Script and Language for them, so\\%
4569 you may consider defining a new family with \string\babelfont.\\%
4570 See the manual for further details about \string\babelfont.\\%
4571 Reported}}
4572 {}%
4573 \gdef\bbl@switchfont{%
4574 \bbl@ifunset{\bbl@lsys@\language name}{\bbl@provide@lsys{\language name}}{}%
4575 \bbl@exp{% eg Arabic -> arabic
4576 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4577 \bbl@foreach\bbl@font@fams{%
4578 \bbl@ifunset{\bbl@##1dflt@\language name}% (1) language?
4579 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4580 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4581 {}% 123=F - nothing!
4582 {\bbl@exp{% 3=T - from generic

```

```

4583 \global\let\<bbl@##1dflt@\language\>%
4584 \<bbl@##1dflt@>}}}%
4585 {\bbl@exp{% 2=T - from script
4586 \global\let\<bbl@##1dflt@\language\>%
4587 \<bbl@##1dflt@*\bbl@tempa>}}}%
4588 {}}}% 1=T - language, already defined
4589 \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4590 \bbl@foreach\bbl@font@fams{% don't gather with prev for
4591 \bbl@ifunset{\bbl@##1dflt@\language\>%
4592 {\bbl@cs{famrst@##1}%
4593 \global\bbl@csarg\let{famrst@##1}\relax}%
4594 {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4595 \\\bbl@add\\\originalTeX{%
4596 \\\bbl@font@rst{\bbl@c\<##1dflt\>}}}%
4597 \<##1default>\<##1family>{##1}}}%
4598 \\\bbl@font@set\<bbl@##1dflt@\language\>% the main part!
4599 \<##1default>\<##1family>}}}%
4600 \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`.

```

4601 \ifx\fbfamily\undefined\else % if latex
4602 \ifcase\bbl@engine % if pdftex
4603 \let\bbl@ckeckstdfonts\relax
4604 \else
4605 \def\bbl@ckeckstdfonts{%
4606 \begin{group}
4607 \global\let\bbl@ckeckstdfonts\relax
4608 \let\bbl@tempa\@empty
4609 \bbl@foreach\bbl@font@fams{%
4610 \bbl@ifunset{\bbl@##1dflt@}%
4611 {\@nameuse{##1family}}%
4612 \bbl@csarg\gdef{WFF@fbfamily}}}% Flag
4613 \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \fbfamily\\}%
4614 \space\space\fontname\font\\}%
4615 \bbl@csarg\xdef{##1dflt@}{fbfamily}%
4616 \expandafter\xdef\csname ##1default\endcsname{\fbfamily}}}%
4617 {}}}%
4618 \ifx\bbl@tempa\@empty\else
4619 \bbl@infowarn{The following font families will use the default\\%
4620 settings for all or some languages:\\%
4621 \bbl@tempa
4622 There is nothing intrinsically wrong with it, but\\%
4623 'babel' will no set Script and Language, which could\\%
4624 be relevant in some languages. If your document uses\\%
4625 these families, consider redefining them with \string\babelfont.\\%
4626 Reported}%
4627 \fi
4628 \end{group}
4629 \fi
4630 \fi

```

Now the macros defining the font with `fontspec`.

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

```

4631 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4632 \bbl@xin@{<>}{#1}%
4633 \ifin@
4634 \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4635 \fi
4636 \bbl@exp{% 'Unprotected' macros return prev values
4637 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4638 \\\bbl@ifsamestring{#2}{fbfamily}}%

```

```

4639      {\#3%
4640      \\\bbl@ifsamestring{\f@series}{\bfdefault}{\bfseries}}}%
4641      \let\\bbl@tempa\relax}%
4642      {}}}
4643 %      TODO - next should be global?, but even local does its job. I'm
4644 %      still not sure -- must investigate:
4645 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4646 \let\bbl@tempe\bbl@mapselect
4647 \let\bbl@mapselect\relax
4648 \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4649 \let#4\empty %      Make sure \renewfontfamily is valid
4650 \bbl@exp{%
4651 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4652 <\keys_if_exist:nnF>{\fontspec-opentype}{Script/\bbl@cl{sname}}}%
4653 {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4654 <\keys_if_exist:nnF>{\fontspec-opentype}{Language/\bbl@cl{lname}}}%
4655 {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4656 \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4657 \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4658 \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4659 \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4660 \\\renewfontfamily\\#4%
4661 [\bbl@cl{lsys},#2]{#3}% ie \bbl@exp{.}{#3}
4662 \bbl@exp{%
4663 \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4664 \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4665 \begingroup
4666 #4%
4667 \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4668 \endgroup
4669 \let#4\bbl@temp@fam
4670 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4671 \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.

```

4672 \def\bbl@font@rst#1#2#3#4{%
4673 \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.

```

4674 \def\bbl@font@fams{rm,sf,tt}
4675 <</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.

```

4676 <<{*Footnote changes}>> ≡
4677 \bbl@trace{Bidi footnotes}
4678 \ifnum\bbl@bidimode>\z@ % Any bidi=
4679 \def\bbl@footnote#1#2#3{%
4680 \@ifnextchar[%
4681 {\bbl@footnote@o{#1}{#2}{#3}}%
4682 {\bbl@footnote@x{#1}{#2}{#3}}}
4683 \long\def\bbl@footnote@x#1#2#3#4{%
4684 \bgroup
4685 \select@language@x{\bbl@main@language}%
4686 \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4687 \egroup}
4688 \long\def\bbl@footnote@o#1#2#3[#4]#5{%

```

```

4689 \bgroup
4690 \select@language@x{\bbl@main@language}%
4691 \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4692 \egroup}
4693 \def\bbl@footnotetext#1#2#3{%
4694 \ifnextchar[%
4695 {\bbl@footnotetext@o{#1}{#2}{#3}}%
4696 {\bbl@footnotetext@x{#1}{#2}{#3}}}
4697 \long\def\bbl@footnotetext@x#1#2#3#4{%
4698 \bgroup
4699 \select@language@x{\bbl@main@language}%
4700 \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4701 \egroup}
4702 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4703 \bgroup
4704 \select@language@x{\bbl@main@language}%
4705 \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4706 \egroup}
4707 \def\BabelFootnote#1#2#3#4{%
4708 \ifx\bbl@fn@footnote\undefined
4709 \let\bbl@fn@footnote\footnote
4710 \fi
4711 \ifx\bbl@fn@footnotetext\undefined
4712 \let\bbl@fn@footnotetext\footnotetext
4713 \fi
4714 \bbl@ifblank{#2}%
4715 {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4716 \@namedef{\bbl@stripslash#1text}%
4717 {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4718 {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}{#3}{#4}}%
4719 \@namedef{\bbl@stripslash#1text}%
4720 {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}}
4721 \fi
4722 <</Footnote changes>>

```

Now, the code.

```

4723 <*\xetex>
4724 \def\BabelStringsDefault{unicode}
4725 \let\xebbl@stop\relax
4726 \AddBabelHook{xetex}{encodedcommands}{%
4727 \def\bbl@tempa{#1}%
4728 \ifx\bbl@tempa\empty
4729 \XeTeXinputencoding"bytes"%
4730 \else
4731 \XeTeXinputencoding"#1"%
4732 \fi
4733 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4734 \AddBabelHook{xetex}{stopcommands}{%
4735 \xebbl@stop
4736 \let\xebbl@stop\relax}
4737 \def\bbl@intraspace#1 #2 #3\@@{%
4738 \bbl@csarg\gdef{xeisp@language}%
4739 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4740 \def\bbl@intrapenalty#1\@@{%
4741 \bbl@csarg\gdef{xeipn@language}%
4742 {\XeTeXlinebreakpenalty #1\relax}}
4743 \def\bbl@provide@intraspace{%
4744 \bbl@xin@{/s}{/\bbl@c{l}nbrk}}%
4745 \ifin@else\bbl@xin@{/c}{/\bbl@c{l}nbrk}}\fi
4746 \ifin@
4747 \bbl@ifunset{bbl@intsp@language}%
4748 {\expandafter\ifx\csname bbl@intsp@language\endcsname\empty\else
4749 \ifx\bbl@KVP@intraspace\@nnil

```

```

4750      \bbl@exp{%
4751          \\bbl@intraspace\bbl@cl{intsp}\\\@}%
4752      \fi
4753      \ifx\bbl@KVP@intrapenalty\@nnil
4754          \bbl@intrapenalty0\@@
4755      \fi
4756  \fi
4757  \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4758      \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4759  \fi
4760  \ifx\bbl@KVP@intrapenalty\@nnil\else
4761      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4762  \fi
4763  \bbl@exp{%
4764      % TODO. Execute only once (but redundant):
4765      \\bbl@add\<extras\language>{%
4766          \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4767          \<bbl@xeisp@\language>%
4768          \<bbl@xeipn@\language>}%
4769      \\bbl@tglobal\<extras\language>%
4770      \\bbl@add\<noextras\language>{%
4771          \XeTeXlinebreaklocale ""}%
4772      \\bbl@tglobal\<noextras\language>}%
4773  \ifx\bbl@ispace\@undefined
4774      \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4775      \ifx\AtBeginDocument\@notprerr
4776          \expandafter\@secondoftwo % to execute right now
4777      \fi
4778      \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4779  \fi}%
4780 \fi}
4781 \ifx\DisableBabelHook\@undefined\endinput\fi
4782 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4783 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4784 \DisableBabelHook{babel-fontspec}
4785 <<Font selection>>
4786 \def\bbl@provide@extra#1{}
4787 </xetex>

```

9.2 Layout

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

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

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

```

4788 (*xetex | texxet)
4789 \providecommand\bbl@provide@intraspace{}
4790 \bbl@trace{Redefinitions for bidi layout}
4791 \def\bbl@sspre@caption{%
4792     \bbl@exp{\everyhbox{\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4793 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4794 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4795 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4796 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4797     \def\hangfrom#1{%
4798         \setbox\@tempboxa\hbox{#{1}}%
4799         \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4800         \noindent\box\@tempboxa}
4801 \def\raggedright{%
4802     \let\\\@centercr
4803     \bbl@startskip\z@skip

```

```

4804 \@rightskip\@flushglue
4805 \bbl@endskip\@rightskip
4806 \parindent\z@
4807 \parfillskip\bbl@startskip}
4808 \def\raggedleft{%
4809 \let\\\@centercr
4810 \bbl@startskip\@flushglue
4811 \bbl@endskip\z@skip
4812 \parindent\z@
4813 \parfillskip\bbl@endskip}
4814 \fi
4815 \IfBabelLayout{lists}
4816 {\bbl@sreplace\list
4817 {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4818 \def\bbl@listleftmargin{%
4819 \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4820 \ifcase\bbl@engine
4821 \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4822 \def\p@enumii{\p@enumii}\theenumii{}\fi
4823 \fi
4824 \bbl@sreplace\@verbatim
4825 {\leftskip\@totalleftmargin}%
4826 {\bbl@startskip\textwidth
4827 \advance\bbl@startskip-\linewidth}%
4828 \bbl@sreplace\@verbatim
4829 {\rightskip\z@skip}%
4830 {\bbl@endskip\z@skip}}%
4831 {}
4832 \IfBabelLayout{contents}
4833 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4834 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4835 {}
4836 \IfBabelLayout{columns}
4837 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4838 \def\bbl@outputbox#1{%
4839 \hb@xt@\textwidth{%
4840 \hskip\columnwidth
4841 \hfil
4842 {\normalcolor\vrule \@width\columnseprule}%
4843 \hfil
4844 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4845 \hskip-\textwidth
4846 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4847 \hskip\columnsep
4848 \hskip\columnwidth}}}%
4849 {}
4850 <<Footnote changes>>
4851 \IfBabelLayout{footnotes}%
4852 {\BabelFootnote\footnote\languagename{}\}%
4853 \BabelFootnote\localfootnote\languagename{}\}%
4854 \BabelFootnote\mainfootnote{}\}%
4855 {}

```

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.

```

4856 \IfBabelLayout{counters*}%
4857 {\bbl@add\bbl@opt@layout{.counters.}%
4858 \AddToHook{shipout/before}{%
4859 \let\bbl@tempa\babelsublr
4860 \let\babelsublr\@firstofone
4861 \let\bbl@save@thepage\thepage
4862 \protected@edef\thepage{\thepage}%
4863 \let\babelsublr\bbl@tempa}%

```



```

4864 \AddToHook{shipout/after}{%
4865 \let\thepage\bbl@save@thepage}}{}
4866 \IfBabelLayout{counters}%
4867 {\let\bbl@latinarabic=\@arabic
4868 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}}%
4869 \let\bbl@asciroman=\@roman
4870 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4871 \let\bbl@asciiRoman=\@Roman
4872 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}{}
4873 \fi % end if layout
4874 \</xetex> \texxtet)

```

9.3 8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff.

```

4875 \<texxtet>
4876 \def\bbl@provide@extra#1{%
4877 % == auto-select encoding ==
4878 \ifx\bbl@encoding@select@off\empty\else
4879 \bbl@ifunset{\bbl@encoding@#1}%
4880 {\def\elt##1{,##1,}%
4881 \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4882 \count@\z@
4883 \bbl@foreach\bbl@tempe{%
4884 \def\bbl@tempd{##1}% Save last declared
4885 \advance\count@\@ne}%
4886 \ifnum\count@>\@ne
4887 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4888 \ifx\bbl@tempa\relax \let\bbl@tempa\empty \fi
4889 \bbl@replace\bbl@tempa{ },{,}%
4890 \global\bbl@csarg\let{encoding@#1}\empty
4891 \bbl@xin@{,\bbl@tempd,},{,\bbl@tempa,}%
4892 \ifin@else % if main encoding included in ini, do nothing
4893 \let\bbl@tempb\relax
4894 \bbl@foreach\bbl@tempa{%
4895 \ifx\bbl@tempb\relax
4896 \bbl@xin@{,##1,},{,\bbl@tempe,}%
4897 \ifin@\def\bbl@tempb{##1}\fi
4898 \fi}%
4899 \ifx\bbl@tempb\relax\else
4900 \bbl@exp{%
4901 \global\<bbl@add>\<bbl@preextras@#1>\<bbl@encoding@#1>}%
4902 \gdef\<bbl@encoding@#1>{%
4903 \\ \babel@save\\ \f@encoding
4904 \\ \bbl@add\\ \originalTeX{\selectfont}%
4905 \\ \fontencoding{\bbl@tempb}%
4906 \\ \selectfont}}%
4907 \fi
4908 \fi
4909 \fi}%
4910 {}%
4911 \fi}
4912 \</texxtet>

```

9.4 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`).

```

4913 \*luatex
4914 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4915 \bbl@trace{Read language.dat}
4916 \ifx\bbl@readstream\@undefined
4917   \csname newread\endcsname\bbl@readstream
4918 \fi
4919 \begingroup
4920   \toks@{}
4921   \count@z@ % 0=start, 1=0th, 2=normal
4922   \def\bbl@process@line#1#2 #3 #4 {%
4923     \ifx=#1%
4924       \bbl@process@synonym{#2}%
4925     \else
4926       \bbl@process@language{#1#2}{#3}{#4}%
4927     \fi
4928     \ignorespaces}
4929   \def\bbl@manylang{%
4930     \ifnum\bbl@last>\@ne
4931       \bbl@info{Non-standard hyphenation setup}%
4932     \fi
4933     \let\bbl@manylang\relax}
4934   \def\bbl@process@language#1#2#3{%
4935     \ifcase\count@
4936       \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4937     \or
4938       \count@\tw@
4939     \fi
4940     \ifnum\count@=\tw@
4941       \expandafter\addlanguage\csname l@#1\endcsname
4942       \language\allocationnumber
4943       \chardef\bbl@last\allocationnumber
4944       \bbl@manylang
4945       \let\bbl@elt\relax
4946       \xdef\bbl@languages{%
4947         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}%
4948       \fi
4949       \the\toks@
4950       \toks@{}}

```

```

4951 \def\bbl@process@synonym@aux#1#2{%
4952   \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4953   \let\bbl@elt\relax
4954   \xdef\bbl@languages{%
4955     \bbl@languages\bbl@elt{#1}{#2}{}}}%
4956 \def\bbl@process@synonym#1{%
4957   \ifcase\count@
4958     \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4959     \or
4960     \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}}%
4961     \else
4962     \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4963     \fi}
4964 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4965   \chardef\l@english\z@
4966   \chardef\l@USenglish\z@
4967   \chardef\bbl@last\z@
4968   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4969   \gdef\bbl@languages{%
4970     \bbl@elt{english}{0}{hyphen.tex}}%
4971     \bbl@elt{USenglish}{0}{}}
4972 \else
4973   \global\let\bbl@languages@format\bbl@languages
4974   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4975     \ifnum#2>\z@\else
4976       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4977       \fi}%
4978   \xdef\bbl@languages{\bbl@languages}%
4979 \fi
4980 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}} % Define flags
4981 \bbl@languages
4982 \openin\bbl@readstream=language.dat
4983 \ifeof\bbl@readstream
4984   \bbl@warning{I couldn't find language.dat. No additional\\%
4985     patterns loaded. Reported}%
4986 \else
4987   \loop
4988     \endlinechar\m@ne
4989     \read\bbl@readstream to \bbl@line
4990     \endlinechar\^^M
4991     \if T\ifeof\bbl@readstream F\fi T\relax
4992     \ifx\bbl@line\empty\else
4993       \edef\bbl@line{\bbl@line\space\space\space}%
4994       \expandafter\bbl@process@line\bbl@line\relax
4995     \fi
4996   \repeat
4997 \fi
4998 \closein\bbl@readstream
4999 \endgroup
5000 \bbl@trace{Macros for reading patterns files}
5001 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
5002 \ifx\babelcatcodetablenum\undefined
5003   \ifx\newcatcodetable\undefined
5004     \def\babelcatcodetablenum{5211}
5005     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5006   \else
5007     \newcatcodetable\babelcatcodetablenum
5008     \newcatcodetable\bbl@pattcodes
5009   \fi
5010 \else
5011   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5012 \fi
5013 \def\bbl@luapatterns#1#2{%

```

```

5014 \bbl@get@enc#1:.\@@@
5015 \setbox\z@\hbox\bgroup
5016 \beginingroup
5017 \savecatcodetable\babelcatcodetablenum\relax
5018 \initcatcodetable\bbl@pattcodes\relax
5019 \catcodetable\bbl@pattcodes\relax
5020 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5021 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
5022 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
5023 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5024 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5025 \catcode`\`=12 \catcode`\'=12 \catcode`\ "=12
5026 \input #1\relax
5027 \catcodetable\babelcatcodetablenum\relax
5028 \endgroup
5029 \def\bbl@tempa{#2}%
5030 \ifx\bbl@tempa\@empty\else
5031 \input #2\relax
5032 \fi
5033 \egroup}%
5034 \def\bbl@patterns@lua#1{%
5035 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5036 \csname l@#1\endcsname
5037 \edef\bbl@tempa{#1}%
5038 \else
5039 \csname l@#1:\f@encoding\endcsname
5040 \edef\bbl@tempa{#1:\f@encoding}%
5041 \fi\relax
5042 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5043 \@ifundefined{bbl@hyphendata@the\language}%
5044 {\def\bbl@elt##1##2##3##4{%
5045 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:0T1...
5046 \def\bbl@tempb{##3}%
5047 \ifx\bbl@tempb\@empty\else % if not a synonymous
5048 \def\bbl@tempc{{##3}{##4}}%
5049 \fi
5050 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5051 \fi}%
5052 \bbl@languages
5053 \@ifundefined{bbl@hyphendata@the\language}%
5054 {\bbl@info{No hyphenation patterns were set for\\
5055 language '\bbl@tempa'. Reported}}%
5056 {\expandafter\expandafter\expandafter\bbl@luapatterns
5057 \csname bbl@hyphendata@the\language\endcsname}}}%
5058 \endinput\fi
5059 % Here ends \ifx\AddBabelHook\@undefined
5060 % A few lines are only read by hyphen.cfg
5061 \ifx\DisableBabelHook\@undefined
5062 \AddBabelHook{luatex}{everylanguage}{%
5063 \def\process@language##1##2##3{%
5064 \def\process@line####1####2 ####3 ####4 {}}}
5065 \AddBabelHook{luatex}{loadpatterns}{%
5066 \input #1\relax
5067 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5068 {{#1}}}%
5069 \AddBabelHook{luatex}{loadexceptions}{%
5070 \input #1\relax
5071 \def\bbl@tempb##1##2{{##1}{#1}}%
5072 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5073 {\expandafter\expandafter\expandafter\bbl@tempb
5074 \csname bbl@hyphendata@the\language\endcsname}}
5075 \endinput\fi
5076 % Here stops reading code for hyphen.cfg

```

```

5077 % The following is read the 2nd time it's loaded
5078 \begingroup % TODO - to a lua file
5079 \catcode`\%=12
5080 \catcode`\'=12
5081 \catcode`\ "=12
5082 \catcode`\:=12
5083 \directlua{
5084   Babel = Babel or {}
5085   function Babel.bytes(line)
5086     return line:gsub(".",
5087       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5088   end
5089   function Babel.begin_process_input()
5090     if luatexbase and luatexbase.add_to_callback then
5091       luatexbase.add_to_callback('process_input_buffer',
5092         Babel.bytes, 'Babel.bytes')
5093     else
5094       Babel.callback = callback.find('process_input_buffer')
5095       callback.register('process_input_buffer', Babel.bytes)
5096     end
5097   end
5098   function Babel.end_process_input ()
5099     if luatexbase and luatexbase.remove_from_callback then
5100       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5101     else
5102       callback.register('process_input_buffer', Babel.callback)
5103     end
5104   end
5105   function Babel.addpatterns(pp, lg)
5106     local lg = lang.new(lg)
5107     local pats = lang.patterns(lg) or ''
5108     lang.clear_patterns(lg)
5109     for p in pp:gmatch('[^%s]+') do
5110       ss = ''
5111       for i in string.utfcharacters(p:gsub('%d', '')) do
5112         ss = ss .. '%d?' .. i
5113       end
5114       ss = ss:gsub('^%d%?%', '%%.') .. '%d?'
5115       ss = ss:gsub('%.%d%?$', '%%.')
5116       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5117       if n == 0 then
5118         tex.sprint(
5119           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5120           .. p .. [[{}]])
5121         pats = pats .. ' ' .. p
5122       else
5123         tex.sprint(
5124           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5125           .. p .. [[{}]])
5126       end
5127     end
5128     lang.patterns(lg, pats)
5129   end
5130   Babel.characters = Babel.characters or {}
5131   Babel.ranges = Babel.ranges or {}
5132   function Babel.hlist_has_bidi(head)
5133     local has_bidi = false
5134     local ranges = Babel.ranges
5135     for item in node.traverse(head) do
5136       if item.id == node.id'glyph' then
5137         local itemchar = item.char
5138         local chardata = Babel.characters[itemchar]
5139         local dir = chardata and chardata.d or nil

```

```

5140         if not dir then
5141             for nn, et in ipairs(ranges) do
5142                 if itemchar < et[1] then
5143                     break
5144                 elseif itemchar <= et[2] then
5145                     dir = et[3]
5146                     break
5147                 end
5148             end
5149         end
5150         if dir and (dir == 'al' or dir == 'r') then
5151             has_bidi = true
5152         end
5153     end
5154 end
5155 return has_bidi
5156 end
5157 function Babel.set_chranges_b (script, chrng)
5158     if chrng == '' then return end
5159     texio.write('Replacing ' .. script .. ' script ranges')
5160     Babel.script_blocks[script] = {}
5161     for s, e in string.gmatch(chrng..' ', '(.)%.%.(.)%s') do
5162         table.insert(
5163             Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5164     end
5165 end
5166 function Babel.discard_sublr(str)
5167     if str:find( [[\string\indexentry]] ) and
5168        str:find( [[\string\babelsublr]] ) then
5169         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5170                        function(m) return m:sub(2,-2) end )
5171     end
5172     return str
5173 end
5174 }
5175 \endgroup
5176 \ifx\newattribute\@undefined\else
5177     \newattribute\bbl@attr@locale
5178     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5179     \AddBabelHook{luatex}{beforeextras}{%
5180         \setattribute\bbl@attr@locale\localeid}
5181 \fi
5182 \def\BabelStringsDefault{unicode}
5183 \let\luabbl@stop\relax
5184 \AddBabelHook{luatex}{encodedcommands}{%
5185     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5186     \ifx\bbl@tempa\bbl@tempb\else
5187         \directlua{Babel.begin_process_input()}%
5188         \def\luabbl@stop{%
5189             \directlua{Babel.end_process_input()}}%
5190     \fi}%
5191 \AddBabelHook{luatex}{stopcommands}{%
5192     \luabbl@stop
5193     \let\luabbl@stop\relax}
5194 \AddBabelHook{luatex}{patterns}{%
5195     \@ifundefined{bbl@hyphendata@the\language}%
5196     { \def\bbl@elt##1##2##3##4{%
5197         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5198         \def\bbl@tempb{##3}%
5199         \ifx\bbl@tempb\@empty\else % if not a synonymous
5200             \def\bbl@tempc{{##3}{##4}}%
5201         \fi
5202         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%

```

```

5203     \fi}%
5204     \bbl@languages
5205     \@ifundefined{bbl@hyphendata@the\language}%
5206     {\bbl@info{No hyphenation patterns were set for\%
5207         language '#2'. Reported}}%
5208     {\expandafter\expandafter\expandafter\bbl@luapatterns
5209       \csname bbl@hyphendata@the\language\endcsname}}}%
5210 \@ifundefined{bbl@patterns@}{}%
5211 \begingroup
5212   \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5213   \ifin@else
5214     \ifx\bbl@patterns@\empty\else
5215       \directlua{ Babel.addpatterns(
5216         [[\bbl@patterns@]], \number\language) }%
5217       \fi
5218       \@ifundefined{bbl@patterns@#1}%
5219       \empty
5220       {\directlua{ Babel.addpatterns(
5221         [[\space\csname bbl@patterns@#1\endcsname]],
5222         \number\language) }}%
5223       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5224     \fi
5225   \endgroup}%
5226 \bbl@exp{%
5227   \bbl@ifunset{bbl@prehc@\languagename}}{%
5228     {\bbl@ifblank{\bbl@cs{prehc@\languagename}}}%
5229     {\prehyphenchar=\bbl@cl{prehc}\relax}}}%

```

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

```

5230 \@onlypreamble\babelpatterns
5231 \AtEndOfPackage{%
5232   \newcommand\babelpatterns[2][\empty]{%
5233     \ifx\bbl@patterns\relax
5234       \let\bbl@patterns@\empty
5235     \fi
5236     \ifx\bbl@pttnlist\empty\else
5237       \bbl@warning{%
5238         You must not intermingle \string\selectlanguage\space and\%
5239         \string\babelpatterns\space or some patterns will not\%
5240         be taken into account. Reported}%
5241       \fi
5242       \ifx\@empty#1%
5243         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5244       \else
5245         \edef\bbl@tempb{\zap@space#1 \empty}%
5246         \bbl@for\bbl@tempa\bbl@tempb{%
5247           \bbl@fixname\bbl@tempa
5248           \bbl@iflanguage\bbl@tempa{%
5249             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5250               \@ifundefined{bbl@patterns@\bbl@tempa}%
5251               \empty
5252               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5253               #2}}}%
5254         \fi}}

```

9.5 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.

```

5255% TODO - to a lua file
5256\directlua{
5257  Babel = Babel or {}
5258  Babel.linebreaking = Babel.linebreaking or {}
5259  Babel.linebreaking.before = {}
5260  Babel.linebreaking.after = {}
5261  Babel.locale = {} % Free to use, indexed by \localeid
5262  function Babel.linebreaking.add_before(func, pos)
5263    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5264    if pos == nil then
5265      table.insert(Babel.linebreaking.before, func)
5266    else
5267      table.insert(Babel.linebreaking.before, pos, func)
5268    end
5269  end
5270  function Babel.linebreaking.add_after(func)
5271    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5272    table.insert(Babel.linebreaking.after, func)
5273  end
5274}
5275\def\bbl@intraspace#1 #2 #3\@{#%
5276  \directlua{
5277    Babel = Babel or {}
5278    Babel.intraspaces = Babel.intraspaces or {}
5279    Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5280      {b = #1, p = #2, m = #3}
5281    Babel.locale_props[\the\localeid].intraspace = %
5282      {b = #1, p = #2, m = #3}
5283  }}
5284\def\bbl@intrapenalty#1\@{#%
5285  \directlua{
5286    Babel = Babel or {}
5287    Babel.intrapenalties = Babel.intrapenalties or {}
5288    Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5289    Babel.locale_props[\the\localeid].intrapenalty = #1
5290  }}
5291\begin{group}
5292\catcode`\%=12
5293\catcode`\^=14
5294\catcode`\'=12
5295\catcode`\~=12
5296\gdef\bbl@seaintraspace{^
5297  \let\bbl@seaintraspace\relax
5298  \directlua{
5299    Babel = Babel or {}
5300    Babel.sea_enabled = true
5301    Babel.sea_ranges = Babel.sea_ranges or {}
5302    function Babel.set_chranges (script, chrng)
5303      local c = 0
5304      for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5305        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5306        c = c + 1
5307      end
5308    end
5309    function Babel.sea_disc_to_space (head)
5310      local sea_ranges = Babel.sea_ranges
5311      local last_char = nil
5312      local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5313      for item in node.traverse(head) do
5314        local i = item.id
5315        if i == node.id'glyph' then
5316          last_char = item
5317        elseif i == 7 and item.subtype == 3 and last_char

```



```

5318         and last_char.char > 0x0C99 then
5319     quad = font.getfont(last_char.font).size
5320     for lg, rg in pairs(sea_ranges) do
5321         if last_char.char > rg[1] and last_char.char < rg[2] then
5322             lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5323             local intraspace = Babel.intraspaces[lg]
5324             local intrapenalty = Babel.intrapenalties[lg]
5325             local n
5326             if intrapenalty ~= 0 then
5327                 n = node.new(14, 0) ^% penalty
5328                 n.penalty = intrapenalty
5329                 node.insert_before(head, item, n)
5330             end
5331             n = node.new(12, 13) ^% (glue, spaceskip)
5332             node.setglue(n, intraspace.b * quad,
5333                 intraspace.p * quad,
5334                 intraspace.m * quad)
5335             node.insert_before(head, item, n)
5336             node.remove(head, item)
5337         end
5338     end
5339 end
5340 end
5341 end
5342 }^^
5343 \bbl@luahyphenate}

```

9.6 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.

```

5344 \catcode`\%=14
5345 \gdef\bbl@cjk intraspace{%
5346     \let\bbl@cjk intraspace\relax
5347     \directlua{
5348         Babel = Babel or {}
5349         require('babel-data-cjk.lua')
5350         Babel.cjk_enabled = true
5351         function Babel.cjk_linebreak(head)
5352             local GLYPH = node.id'glyph'
5353             local last_char = nil
5354             local quad = 655360 % 10 pt = 655360 = 10 * 65536
5355             local last_class = nil
5356             local last_lang = nil
5357
5358             for item in node.traverse(head) do
5359                 if item.id == GLYPH then
5360
5361                     local lang = item.lang
5362
5363                     local LOCALE = node.get_attribute(item,
5364                         Babel.attr_locale)
5365                     local props = Babel.locale_props[LOCALE]
5366
5367                     local class = Babel.cjk_class[item.char].c
5368
5369                     if props.cjk_quotes and props.cjk_quotes[item.char] then
5370                         class = props.cjk_quotes[item.char]
5371                     end

```

```

5372
5373     if class == 'cp' then class = 'cl' end % ]] as CL
5374     if class == 'id' then class = 'I' end
5375
5376     local br = 0
5377     if class and last_class and Babel.cjk_breaks[last_class][class] then
5378         br = Babel.cjk_breaks[last_class][class]
5379     end
5380
5381     if br == 1 and props.linebreak == 'c' and
5382         lang ~= \the\l@nohyphenation\space and
5383         last_lang ~= \the\l@nohyphenation then
5384         local intrapenalty = props.intrapenalty
5385         if intrapenalty ~= 0 then
5386             local n = node.new(14, 0)      % penalty
5387             n.penalty = intrapenalty
5388             node.insert_before(head, item, n)
5389         end
5390         local intraspace = props.intraspace
5391         local n = node.new(12, 13)        % (glue, spaceskip)
5392         node.setglue(n, intraspace.b * quad,
5393             intraspace.p * quad,
5394             intraspace.m * quad)
5395         node.insert_before(head, item, n)
5396     end
5397
5398     if font.getfont(item.font) then
5399         quad = font.getfont(item.font).size
5400     end
5401     last_class = class
5402     last_lang = lang
5403     else % if penalty, glue or anything else
5404         last_class = nil
5405     end
5406 end
5407 lang.hyphenate(head)
5408 end
5409 }%
5410 \bbl@luahyphenate}
5411 \gdef\bbl@luahyphenate{%
5412 \let\bbl@luahyphenate\relax
5413 \directlua{
5414     luatexbase.add_to_callback('hyphenate',
5415     function (head, tail)
5416         if Babel.linebreaking.before then
5417             for k, func in ipairs(Babel.linebreaking.before) do
5418                 func(head)
5419             end
5420         end
5421         if Babel.cjk_enabled then
5422             Babel.cjk_linebreak(head)
5423         end
5424         lang.hyphenate(head)
5425         if Babel.linebreaking.after then
5426             for k, func in ipairs(Babel.linebreaking.after) do
5427                 func(head)
5428             end
5429         end
5430         if Babel.sea_enabled then
5431             Babel.sea_disc_to_space(head)
5432         end
5433     end,
5434     'Babel.hyphenate')

```

```

5435 }
5436 }
5437 \endgroup
5438 \def\bbbl@provide@intraspace{%
5439   \bbbl@ifunset{\bbbl@intsp@{language}\language}{}%
5440   {\expandafter\ifx\csname \bbbl@intsp@{language}\endcsname\@empty\else
5441     \bbbl@xin@{/c}{/\bbbl@cl{lnbrk}}%
5442     \ifin@           % cjk
5443     \bbbl@cjk@intraspace
5444     \directlua{
5445       Babel = Babel or {}
5446       Babel.locale_props = Babel.locale_props or {}
5447       Babel.locale_props[\the\localeid].linebreak = 'c'
5448     }%
5449     \bbbl@exp{\bbbl@intraspace\bbbl@cl{intsp}}\@@}%
5450     \ifx\bbbl@KVP@intrapenalty\@nnil
5451       \bbbl@intrapenalty0\@@
5452     \fi
5453   \else           % sea
5454     \bbbl@sea@intraspace
5455     \bbbl@exp{\bbbl@intraspace\bbbl@cl{intsp}}\@@}%
5456     \directlua{
5457       Babel = Babel or {}
5458       Babel.sea_ranges = Babel.sea_ranges or {}
5459       Babel.set_chranges('\bbbl@cl{sbcpr}',
5460                           '\bbbl@cl{chrng}')
5461     }%
5462     \ifx\bbbl@KVP@intrapenalty\@nnil
5463       \bbbl@intrapenalty0\@@
5464     \fi
5465   \fi
5466   \fi
5467   \ifx\bbbl@KVP@intrapenalty\@nnil\else
5468     \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty\@@
5469   \fi}}

```

9.7 Arabic justification

```

5470 \ifnum\bbbl@bidimode>100 \ifnum\bbbl@bidimode<200
5471 \def\bbblar@chars{%
5472   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5473   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5474   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5475 \def\bbblar@elongated{%
5476   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5477   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5478   0649,064A}
5479 \begingroup
5480   \catcode`\_ =11 \catcode`\:=11
5481   \gdef\bbblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5482 \endgroup
5483 \gdef\bbblar@arabicjust{%
5484   \let\bbblar@arabicjust\relax
5485   \newattribute\bbblar@kashida
5486   \newattribute\bbblar@kashida@aux % 0, 1=tatweel, 2=diacritics
5487   \directlua{% WIP
5488     Babel.attr_kashida = luatexbase.registernumber'bbblar@kashida'
5489     Babel.attr_kashida_aux = luatexbase.registernumber'bbblar@kashida@aux'
5490   }%
5491   \bbblar@kashida=\z@
5492   \bbblar@kashida@aux=\z@
5493   \bbbl@patchfont{\bbbl@parseja\l}%
5494   \directlua{

```

```

5495 Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5496 Babel.arabic.elong_map[\the\localeid] = {}
5497 luatexbase.add_to_callback('post_linebreak_filter',
5498   Babel.arabic.justify, 'Babel.arabic.justify')
5499 luatexbase.add_to_callback('hpack_filter',
5500   Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5501 }%
5502 % Save both node lists to make replacement. TODO. Save also widths to
5503 % make computations
5504 \def\bblar@fetchjalt#1#2#3#4{%
5505   \bbl@exp{\bbl@foreach{#1}}{%
5506     \bbl@ifunset{bblar@JE@##1}%
5507       {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5508       {\setbox\z@\hbox{^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5509     \directlua{%
5510       local last = nil
5511       for item in node.traverse(tex.box[0].head) do
5512         if item.id == node.id'glyph' and item.char > 0x600 and
5513           not (item.char == 0x200D) then
5514           last = item
5515         end
5516       end
5517       Babel.arabic.#3['##1#4'] = last.char
5518     }}
5519 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5520 % perhaps other tables (falt?, csw?). What about kaf? And diacritic
5521 % positioning?
5522 \gdef\bbl@parsejalt{%
5523   \ifx\addfontfeature\@undefined\else
5524     \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
5525     \ifin@
5526       \directlua{%
5527         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5528           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5529           tex.print([\string\curname\space bbl@parsejalti\endcurname])
5530         end
5531       }%
5532     \fi
5533   \fi}
5534 \gdef\bbl@parsejalti{%
5535   \begingroup
5536     \let\bbl@parsejalt\relax % To avoid infinite loop
5537     \edef\bbl@tempb{\fontid\font}%
5538     \bblar@nofswarn
5539     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5540     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5541     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5542     \addfontfeature{RawFeature+=jalt}%
5543     % \namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5544     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5545     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5546     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5547     \directlua{%
5548       for k, v in pairs(Babel.arabic.from) do
5549         if Babel.arabic.dest[k] and
5550           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5551           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5552             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5553         end
5554       end
5555     }%
5556   \endgroup}
5557 %

```

```

5558 \beginingroup
5559 \catcode`#=11
5560 \catcode`~=11
5561 \directlua{
5562
5563 Babel.arabic = Babel.arabic or {}
5564 Babel.arabic.from = {}
5565 Babel.arabic.dest = {}
5566 Babel.arabic.justify_factor = 0.95
5567 Babel.arabic.justify_enabled = true
5568 Babel.arabic.tatwil_max = -1
5569
5570 function Babel.arabic.justify(head)
5571   if not Babel.arabic.justify_enabled then return head end
5572   for line in node.traverse_id(node.id'hlist', head) do
5573     Babel.arabic.justify_hlist(head, line)
5574   end
5575   return head
5576 end
5577
5578 function Babel.arabic.justify_hbox(head, gc, size, pack)
5579   local has_inf = false
5580   if Babel.arabic.justify_enabled and pack == 'exactly' then
5581     for n in node.traverse_id(12, head) do
5582       if n.stretch_order > 0 then has_inf = true end
5583     end
5584     if not has_inf then
5585       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5586     end
5587   end
5588   return head
5589 end
5590
5591 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5592   local d, new
5593   local k_list, k_item, pos_inline
5594   local width, width_new, full, k_curr, wt_pos, goal, shift
5595   local subst_done = false
5596   local elong_map = Babel.arabic.elong_map
5597   local cnt
5598   local last_line
5599   local GLYPH = node.id'glyph'
5600   local KASHIDA = Babel.attr_kashida
5601   local LOCALE = Babel.attr_locale
5602
5603   if line == nil then
5604     line = {}
5605     line.glue_sign = 1
5606     line.glue_order = 0
5607     line.head = head
5608     line.shift = 0
5609     line.width = size
5610   end
5611
5612   % Exclude last line. todo. But-- it discards one-word lines, too!
5613   % ? Look for glue = 12:15
5614   if (line.glue_sign == 1 and line.glue_order == 0) then
5615     elongs = {} % Stores elongated candidates of each line
5616     k_list = {} % And all letters with kashida
5617     pos_inline = 0 % Not yet used
5618
5619     for n in node.traverse_id(GLYPH, line.head) do
5620       pos_inline = pos_inline + 1 % To find where it is. Not used.

```

```

5621
5622 % Elongated glyphs
5623 if elong_map then
5624     local locale = node.get_attribute(n, LOCALE)
5625     if elong_map[locale] and elong_map[locale][n.font] and
5626         elong_map[locale][n.font][n.char] then
5627         table.insert(elongs, {node = n, locale = locale} )
5628         node.set_attribute(n.prev, KASHIDA, 0)
5629     end
5630 end
5631
5632 % Tatwil
5633 if Babel.kashida_wts then
5634     local k_wt = node.get_attribute(n, KASHIDA)
5635     if k_wt > 0 then % todo. parameter for multi inserts
5636         table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5637     end
5638 end
5639
5640 end % of node.traverse_id
5641
5642 if #elongs == 0 and #k_list == 0 then goto next_line end
5643 full = line.width
5644 shift = line.shift
5645 goal = full * Babel.arabic.justify_factor % A bit crude
5646 width = node.dimensions(line.head) % The 'natural' width
5647
5648 % == Elongated ==
5649 % Original idea taken from 'chickenize'
5650 while (#elongs > 0 and width < goal) do
5651     subst_done = true
5652     local x = #elongs
5653     local curr = elongs[x].node
5654     local oldchar = curr.char
5655     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5656     width = node.dimensions(line.head) % Check if the line is too wide
5657     % Substitute back if the line would be too wide and break:
5658     if width > goal then
5659         curr.char = oldchar
5660         break
5661     end
5662     % If continue, pop the just substituted node from the list:
5663     table.remove(elongs, x)
5664 end
5665
5666 % == Tatwil ==
5667 if #k_list == 0 then goto next_line end
5668
5669 width = node.dimensions(line.head) % The 'natural' width
5670 k_curr = #k_list % Traverse backwards, from the end
5671 wt_pos = 1
5672
5673 while width < goal do
5674     subst_done = true
5675     k_item = k_list[k_curr].node
5676     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5677         d = node.copy(k_item)
5678         d.char = 0x0640
5679         line.head, new = node.insert_after(line.head, k_item, d)
5680         width_new = node.dimensions(line.head)
5681         if width > goal or width == width_new then
5682             node.remove(line.head, new) % Better compute before
5683             break

```

```

5684         end
5685         width = width_new
5686     end
5687     if k_curr == 1 then
5688         k_curr = #k_list
5689         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5690     else
5691         k_curr = k_curr - 1
5692     end
5693 end
5694
5695 % Limit the number of tatweel by removing them. Not very efficient,
5696 % but it does the job in a quite predictable way.
5697 if Babel.arabic.tatwil_max > -1 then
5698     cnt = 0
5699     for n in node.traverse_id(GLYPH, line.head) do
5700         if n.char == 0x0640 then
5701             cnt = cnt + 1
5702             if cnt > Babel.arabic.tatwil_max then
5703                 node.remove(line.head, n)
5704             end
5705         else
5706             cnt = 0
5707         end
5708     end
5709 end
5710
5711 ::next_line::
5712
5713 % Must take into account marks and ins, see luatex manual.
5714 % Have to be executed only if there are changes. Investigate
5715 % what's going on exactly.
5716 if subst_done and not gc then
5717     d = node.hpack(line.head, full, 'exactly')
5718     d.shift = shift
5719     node.insert_before(head, line, d)
5720     node.remove(head, line)
5721 end
5722 end % if process line
5723 end
5724 }
5725 \endgroup
5726 \fi\fi % Arabic just block

```

9.8 Common stuff

```

5727 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5728 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5729 \DisableBabelHook{babel-fontspec}
5730 <<Font selection>>

```

9.9 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 short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` 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.

```

5731 % TODO - to a lua file
5732 \directlua{
5733 Babel.script_blocks = {
5734   ['dflt'] = {},

```

```

5735 ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5736           {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5737 ['Armn'] = {{0x0530, 0x058F}},
5738 ['Beng'] = {{0x0980, 0x09FF}},
5739 ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5740 ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5741 ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5742           {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5743 ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5744 ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5745           {0xAB00, 0xAB2F}},
5746 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5747 % Don't follow strictly Unicode, which places some Coptic letters in
5748 % the 'Greek and Coptic' block
5749 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5750 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5751           {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5752           {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF}},
5753           {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5754           {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5755           {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5756 ['Hebr'] = {{0x0590, 0x05FF}},
5757 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5758           {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5759 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5760 ['Knda'] = {{0x0C80, 0x0CFF}},
5761 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5762           {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5763           {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5764 ['Lao'] = {{0x0E80, 0x0EFF}},
5765 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5766           {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5767           {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5768 ['Mahj'] = {{0x1150, 0x117F}},
5769 ['Mlym'] = {{0x0D00, 0x0D7F}},
5770 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5771 ['Orya'] = {{0x0B00, 0x0B7F}},
5772 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5773 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5774 ['Taml'] = {{0x0B80, 0x0BFF}},
5775 ['Telu'] = {{0x0C00, 0x0C7F}},
5776 ['Tfng'] = {{0x2D30, 0x2D7F}},
5777 ['Thai'] = {{0x0E00, 0x0E7F}},
5778 ['Tibt'] = {{0x0F00, 0x0FFF}},
5779 ['Vaii'] = {{0xA500, 0xA63F}},
5780 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5781 }
5782
5783 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyril
5784 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5785 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5786
5787 function Babel.locale_map(head)
5788   if not Babel.locale_mapped then return head end
5789
5790   local LOCALE = Babel.attr_locale
5791   local GLYPH = node.id('glyph')
5792   local inmath = false
5793   local toloc_save
5794   for item in node.traverse(head) do
5795     local toloc
5796     if not inmath and item.id == GLYPH then
5797       % Optimization: build a table with the chars found

```



```

5798     if Babel.chr_to_loc[item.char] then
5799         toloc = Babel.chr_to_loc[item.char]
5800     else
5801         for lc, maps in pairs(Babel.loc_to_scr) do
5802             for _, rg in pairs(maps) do
5803                 if item.char >= rg[1] and item.char <= rg[2] then
5804                     Babel.chr_to_loc[item.char] = lc
5805                     toloc = lc
5806                     break
5807                 end
5808             end
5809         end
5810     end
5811     % Now, take action, but treat composite chars in a different
5812     % fashion, because they 'inherit' the previous locale. Not yet
5813     % optimized.
5814     if not toloc and
5815         (item.char >= 0x0300 and item.char <= 0x036F) or
5816         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5817         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5818         toloc = toloc_save
5819     end
5820     if toloc and Babel.locale_props[toloc] and
5821         Babel.locale_props[toloc].letters and
5822         tex.getcatcode(item.char) \string~= 11 then
5823         toloc = nil
5824     end
5825     if toloc and toloc > -1 then
5826         if Babel.locale_props[toloc].lg then
5827             item.lang = Babel.locale_props[toloc].lg
5828             node.set_attribute(item, LOCALE, toloc)
5829         end
5830         if Babel.locale_props[toloc]['/'..item.font] then
5831             item.font = Babel.locale_props[toloc]['/'..item.font]
5832         end
5833         toloc_save = toloc
5834     end
5835     elseif not inmath and item.id == 7 then % Apply recursively
5836         item.replace = item.replace and Babel.locale_map(item.replace)
5837         item.pre = item.pre and Babel.locale_map(item.pre)
5838         item.post = item.post and Babel.locale_map(item.post)
5839     elseif item.id == node.id'math' then
5840         inmath = (item.subtype == 0)
5841     end
5842 end
5843 return head
5844 end
5845 }

```

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

```

5846 \newcommand\babelcharproperty[1]{%
5847   \count@=#1\relax
5848   \ifvmode
5849     \expandafter\bbl@chprop
5850   \else
5851     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5852               vertical mode (preamble or between paragraphs)}%
5853     {See the manual for futher info}%
5854   \fi}
5855 \newcommand\bbl@chprop[3][\the\count@]{%
5856   \@tempcnta=#1\relax
5857   \bbl@ifunset{\bbl@chprop@#2}%

```

```

5858     {\bbl@error{No property named '#2'. Allowed values are\%
5859         direction (bc), mirror (bmg), and linebreak (lb))}%
5860         {See the manual for futher info}}}%
5861     }%
5862     \loop
5863       \bbl@cs{chprop@#2}{#3}%
5864       \ifnum\count@<\@tempcnta
5865         \advance\count@\@ne
5866       \repeat}
5867 \def\bbl@chprop@direction#1{%
5868   \directlua{
5869     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5870     Babel.characters[\the\count@]['d'] = '#1'
5871   }}
5872 \let\bbl@chprop@bc\bbl@chprop@direction
5873 \def\bbl@chprop@mirror#1{%
5874   \directlua{
5875     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5876     Babel.characters[\the\count@]['m'] = '\number#1'
5877   }}
5878 \let\bbl@chprop@bmg\bbl@chprop@mirror
5879 \def\bbl@chprop@linebreak#1{%
5880   \directlua{
5881     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5882     Babel.cjk_characters[\the\count@]['c'] = '#1'
5883   }}
5884 \let\bbl@chprop@lb\bbl@chprop@linebreak
5885 \def\bbl@chprop@locale#1{%
5886   \directlua{
5887     Babel.chr_to_loc = Babel.chr_to_loc or {}
5888     Babel.chr_to_loc[\the\count@] =
5889       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5890   }}

```

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.

```

5891 \directlua{
5892   Babel.nohyphenation = \the\l@nohyphenation
5893 }

```

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

```

5894 \begingroup
5895 \catcode`\~ = 12
5896 \catcode`\% = 12
5897 \catcode`\& = 14
5898 \catcode`\| = 12
5899 \gdef\babelprehyphenation{%&
5900   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}}{}]}
5901 \gdef\babelposthyphenation{%&
5902   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}}{}]}
5903 \gdef\bbl@postlinebreak{\bbl@settransform{2}}{} &% WIP
5904 \gdef\bbl@settransform#1[#2]#3#4#5{%&
5905   \ifcase#1
5906     \bbl@activateprehyphen
5907   \or
5908     \bbl@activateposthyphen

```

```

5909 \fi
5910 \beginingroup
5911 \def\babeltempa{\bbl@add@list\babeltempb}&%
5912 \let\babeltempb@empty
5913 \def\bbl@tempa{#5}&%
5914 \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5915 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5916 \bbl@ifsamestring{##1}{remove}&%
5917 {\bbl@add@list\babeltempb{nil}}&%
5918 {\directlua{
5919 local rep = {[##1]=}
5920 rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5921 rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5922 rep = rep:gsub('(string)%s*=%s*([%s,]*)', Babel.capture_func)
5923 if #1 == 0 or #1 == 2 then
5924 rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5925 'space = {' .. '%2, %3, %4' .. '}')
5926 rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5927 'spacefactor = {' .. '%2, %3, %4' .. '}')
5928 rep = rep:gsub('(kashida)%s*=%s*([%s,]*)', Babel.capture_kashida)
5929 else
5930 rep = rep:gsub(' (no)%s*=%s*([%s,]*)', Babel.capture_func)
5931 rep = rep:gsub(' (pre)%s*=%s*([%s,]*)', Babel.capture_func)
5932 rep = rep:gsub(' (post)%s*=%s*([%s,]*)', Babel.capture_func)
5933 end
5934 tex.print([[string\babeltempa{[]] .. rep .. [{}]])
5935 ]}}&%
5936 \bbl@foreach\babeltempb{&%
5937 \bbl@forkv{##1}}{&%
5938 \in@{,###1,}{,nil,step,data,remove,insert,string,no,pre,&%
5939 no,post,penalty,kashida,space,spacefactor,&%
5940 \ifin@}\else
5941 \bbl@error
5942 {Bad option '###1' in a transform.\\&%
5943 I'll ignore it but expect more errors}&%
5944 {See the manual for further info.}&%
5945 \fi}}&%
5946 \let\bbl@kv@attribute\relax
5947 \let\bbl@kv@label\relax
5948 \let\bbl@kv@fonts@empty
5949 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5950 \ifx\bbl@kv@fonts@empty\else\bbl@settransfont\fi
5951 \ifx\bbl@kv@attribute\relax
5952 \ifx\bbl@kv@label\relax\else
5953 \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
5954 \bbl@replace\bbl@kv@fonts{ },}&%
5955 \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
5956 \count@ \z@
5957 \def\bbl@elt##1##2##3{&%
5958 \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
5959 {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5960 {\count@ \one}&%
5961 {\bbl@error
5962 {Transforms cannot be re-assigned to different\\&%
5963 fonts. The conflict is in '\bbl@kv@label'.\\&%
5964 Apply the same fonts or use a different label}&%
5965 {See the manual for further details.}}}&%
5966 }}}&%
5967 \bbl@transfont@list
5968 \ifnum\count@=\z@
5969 \bbl@exp{\global\bbl@add\bbl@transfont@list
5970 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
5971 \fi

```

```

5972     \bbl@ifunset{\bbl@kv@attribute}&%
5973     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
5974     {}&%
5975     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
5976   \fi
5977 \else
5978   \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5979 \fi
5980 \directlua{
5981   local lbkr = Babel.linebreaking.replacements[#1]
5982   local u = unicode.utf8
5983   local id, attr, label
5984   if #1 == 0 or #1 == 2 then
5985     id = \the\csname bbl@id@#3\endcsname\space
5986   else
5987     id = \the\csname l@#3\endcsname\space
5988   end
5989   \ifx\bbl@kv@attribute\relax
5990     attr = -1
5991   \else
5992     attr = luatexbase.registernumber'\bbl@kv@attribute'
5993   \fi
5994   \ifx\bbl@kv@label\relax\else &% Same refs:
5995     label = [==[\bbl@kv@label]==]
5996   \fi
5997   &% Convert pattern:
5998   local patt = string.gsub([==[#4]==], '%s', '')
5999   if #1 == 0 or #1 == 2 then
6000     patt = string.gsub(patt, '|', ' ')
6001   end
6002   if not u.find(patt, '()', nil, true) then
6003     patt = '()' .. patt .. '()'
6004   end
6005   if #1 == 1 then
6006     patt = string.gsub(patt, '%(%)%^', '^()')
6007     patt = string.gsub(patt, '%$(%)', '()$')
6008   end
6009   patt = u.gsub(patt, '{(.)}',
6010     function (n)
6011       return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6012     end)
6013   patt = u.gsub(patt, '{(%x%x%x%x+)}',
6014     function (n)
6015       return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6016     end)
6017   lbkr[id] = lbkr[id] or {}
6018   table.insert(lbkr[id],
6019     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6020   }&%
6021 \endgroup}
6022 \endgroup
6023 \let\bbl@transfont@list\@empty
6024 \def\bbl@settransfont{%
6025   \global\let\bbl@settransfont\relax % Execute only once
6026   \gdef\bbl@transfont{%
6027     \def\bbl@elt####1####2####3{%
6028       \bbl@ifblank{####3}%
6029       {\count@tw@}% Do nothing if no fonts
6030       {\count@z@
6031         \bbl@vforeach{####3}{%
6032           \def\bbl@tempd{#####1}%
6033           \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6034           \ifx\bbl@tempd\bbl@tempe

```

```

6035         \count@\@ne
6036     \elseifx\bbbl@tempd\bbbl@transfam
6037         \count@\@ne
6038     \fi\fi}%
6039 \ifcase\count@
6040     \bbbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6041 \or
6042     \bbbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6043 \fi}%
6044 \bbbl@transfont@list}%
6045 \AddToHook{selectfont}{\bbbl@transfont}% Hooks are global.
6046 \gdef\bbbl@transfam{-unknown-}%
6047 \bbbl@foreach\bbbl@font@fams{%
6048     \AddToHook{##1family}{\def\bbbl@transfam{##1}}%
6049     \bbbl@ifsamestring{\@nameuse{##1default}}\familydefault
6050     {\xdef\bbbl@transfam{##1}}%
6051     {}}%
6052 \DeclareRobustCommand\enablelocaletransform[1]{%
6053     \bbbl@ifunset{\bbbl@ATR@#1@\language @}%
6054     {\bbbl@error
6055         {'#1' for '\language' cannot be enabled.\\%
6056         Maybe there is a typo or it's a font-dependent transform}%
6057         {See the manual for further details.}}%
6058     {\bbbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6059 \DeclareRobustCommand\disablelocaletransform[1]{%
6060     \bbbl@ifunset{\bbbl@ATR@#1@\language @}%
6061     {\bbbl@error
6062         {'#1' for '\language' cannot be disabled.\\%
6063         Maybe there is a typo or it's a font-dependent transform}%
6064         {See the manual for further details.}}%
6065     {\bbbl@csarg\unsetattribute{ATR@#1@\language @}}}
6066 \def\bbbl@activateposthyphen{%
6067     \let\bbbl@activateposthyphen\relax
6068     \directlua{
6069         require('babel-transforms.lua')
6070         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6071     }}
6072 \def\bbbl@activateprehyphen{%
6073     \let\bbbl@activateprehyphen\relax
6074     \directlua{
6075         require('babel-transforms.lua')
6076         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6077     }}

```

9.10 Bidi

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

```

6078 \def\bbbl@activate@preotf{%
6079     \let\bbbl@activate@preotf\relax % only once
6080     \directlua{
6081         Babel = Babel or {}
6082         %
6083         function Babel.pre_otfload_v(head)
6084             if Babel.numbers and Babel.digits_mapped then
6085                 head = Babel.numbers(head)
6086             end
6087             if Babel.bidi_enabled then
6088                 head = Babel.bidi(head, false, dir)
6089             end
6090             return head
6091         end

```

```

6092 %
6093 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6094   if Babel.numbers and Babel.digits_mapped then
6095     head = Babel.numbers(head)
6096   end
6097   if Babel.bidi_enabled then
6098     head = Babel.bidi(head, false, dir)
6099   end
6100   return head
6101 end
6102 %
6103 luatexbase.add_to_callback('pre_linebreak_filter',
6104   Babel.pre_otfload_v,
6105   'Babel.pre_otfload_v',
6106   luatexbase.priority_in_callback('pre_linebreak_filter',
6107     'luaotfload.node_processor') or nil)
6108 %
6109 luatexbase.add_to_callback('hpack_filter',
6110   Babel.pre_otfload_h,
6111   'Babel.pre_otfload_h',
6112   luatexbase.priority_in_callback('hpack_filter',
6113     'luaotfload.node_processor') or nil)
6114 }}

```

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=`.

```

6115 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6116 \let\bbl@beforeforeign\leavevmode
6117 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6118 \RequirePackage{luatexbase}
6119 \bbl@activate@preotf
6120 \directlua{
6121   require('babel-data-bidi.lua')
6122   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6123     require('babel-bidi-basic.lua')
6124   \or
6125     require('babel-bidi-basic-r.lua')
6126   \fi}
6127 \newattribute\bbl@attr@dir
6128 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6129 \bbl@exp{\output{\bodydir\pagedir\the\output}}
6130 \fi
6131 \chardef\bbl@thetextdir\z@
6132 \chardef\bbl@thepardir\z@
6133 \def\bbl@getluadir#1{%
6134   \directlua{
6135     if tex.#ldir == 'TLT' then
6136       tex.sprint('0')
6137     elseif tex.#ldir == 'TRT' then
6138       tex.sprint('1')
6139     end}}
6140 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6141   \ifcase#3\relax
6142     \ifcase\bbl@getluadir{#1}\relax\else
6143       #2 TLT\relax
6144     \fi
6145   \else
6146     \ifcase\bbl@getluadir{#1}\relax
6147       #2 TRT\relax
6148     \fi
6149   \fi}
6150 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)

```

```

6151 \def\bbl@thedir{0}
6152 \def\bbl@textdir#1{%
6153   \bbl@setluadir{text}\textdir{#1}%
6154   \chardef\bbl@thetextdir#1\relax
6155   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6156   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6157 \def\bbl@pardir#1{% Used twice
6158   \bbl@setluadir{par}\pardir{#1}%
6159   \chardef\bbl@thepardir#1\relax}
6160 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6161 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6162 \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.

```

6163 \ifnum\bbl@bidimode>\z@ % Any bidi=
6164   \def\bbl@insidemath{0}%
6165   \def\bbl@everymath{\def\bbl@insidemath{1}}
6166   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6167   \frozen@everymath\expandafter{%
6168     \expandafter\bbl@everymath\the\frozen@everymath}
6169   \frozen@everydisplay\expandafter{%
6170     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6171   \AtBeginDocument{
6172     \directlua{
6173       function Babel.math_box_dir(head)
6174         if not (token.get_macro('bbl@insidemath') == '0') then
6175           if Babel.hlist_has_bidi(head) then
6176             local d = node.new(node.id'dir')
6177             d.dir = '+TRT'
6178             node.insert_before(head, node.has_glyph(head), d)
6179             for item in node.traverse(head) do
6180               node.set_attribute(item,
6181                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6182             end
6183           end
6184         end
6185         return head
6186       end
6187       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6188         "Babel.math_box_dir", 0)
6189     }}%
6190 \fi

```

9.11 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.

```

6191 \bbl@trace{Redefinitions for bidi layout}
6192 %
6193 <<{*More package options}>> ≡
6194 \chardef\bbl@eqnpos\z@
6195 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6196 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6197 <</More package options>>
6198 %
6199 \ifnum\bbl@bidimode>\z@ % Any bidi=
6200   \ifx\matheqdirmode\undefined\else
6201     \matheqdirmode\@ne % A luatex primitive
6202   \fi
6203   \let\bbl@eqnodir\relax
6204   \def\bbl@eqdel{()}
6205   \def\bbl@eqnum{%
6206     {\normalfont\normalcolor
6207       \expandafter\@firstoftwo\bbl@eqdel
6208       \theequation
6209       \expandafter\@secondoftwo\bbl@eqdel}}
6210   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6211   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6212   \def\bbl@eqno@flip#1{%
6213     \ifdim\predisplaysize=-\maxdimen
6214       \eqno
6215       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6216     \else
6217       \leqno\hbox{#1}%
6218     \fi}
6219   \def\bbl@leqno@flip#1{%
6220     \ifdim\predisplaysize=-\maxdimen
6221       \leqno
6222       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6223     \else
6224       \eqno\hbox{#1}%
6225     \fi}
6226   \AtBeginDocument{%
6227     \ifx\bbl@noamsmath\relax\else
6228       \ifx\maketag@@@\undefined % Normal equation, eqnarray
6229         \AddToHook{env/equation/begin}{%
6230           \ifnum\bbl@thetextdir>\z@
6231             \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6232             \let\@eqnnum\bbl@eqnum
6233             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6234             \chardef\bbl@thetextdir\z@
6235             \bbl@add\normalfont{\bbl@eqnodir}%
6236             \ifcase\bbl@eqnpos
6237               \let\bbl@puteqno\bbl@eqno@flip
6238             \or
6239               \let\bbl@puteqno\bbl@leqno@flip
6240             \fi
6241           \fi}%
6242         \ifnum\bbl@eqnpos=\tw@\else
6243           \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6244         \fi
6245         \AddToHook{env/eqnarray/begin}{%
6246           \ifnum\bbl@thetextdir>\z@
6247             \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6248             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6249             \chardef\bbl@thetextdir\z@
6250             \bbl@add\normalfont{\bbl@eqnodir}%
6251           \ifnum\bbl@eqnpos=\@ne

```



```

6252         \def\@eqnnum{%
6253             \setbox\z@\hbox{\bbl@eqnum}%
6254             \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6255         \else
6256             \let\@eqnnum\bbl@eqnum
6257         \fi
6258     \fi}
6259 % Hack. YA luatex bug?:
6260 \expandafter\bbl@sreplace\csname] \endcsname{${$}\eqno\kern.001pt${$}%
6261 \else % amstex
6262     \bbl@exp{% Hack to hide maybe undefined conditionals:
6263         \chardef\bbl@eqnpos=0%
6264         \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6265     \ifnum\bbl@eqnpos=\@ne
6266         \let\bbl@ams@lap\hbox
6267     \else
6268         \let\bbl@ams@lap\llap
6269     \fi
6270 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6271 \bbl@sreplace\intertext@{\normalbaselines}%
6272     {\normalbaselines
6273     \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6274 \ExplSyntaxOff
6275 \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6276 \ifx\bbl@ams@lap\hbox % leqno
6277     \def\bbl@ams@flip#1{%
6278         \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6279 \else % eqno
6280     \def\bbl@ams@flip#1{%
6281         \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6282     \fi
6283 \def\bbl@ams@preset#1{%
6284     \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6285     \ifnum\bbl@thetextdir>\z@
6286         \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6287         \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6288         \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6289     \fi}%
6290 \ifnum\bbl@eqnpos=\tw@ \else
6291     \def\bbl@ams@equation{%
6292         \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6293         \ifnum\bbl@thetextdir>\z@
6294             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6295             \chardef\bbl@thetextdir\z@
6296             \bbl@add\normalfont{\bbl@eqnodir}%
6297             \ifcase\bbl@eqnpos
6298                 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6299             \or
6300                 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6301             \fi
6302         \fi}%
6303     \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6304     \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6305 \fi
6306 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6307 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6308 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6309 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6310 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6311 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6312 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6313 % Hackish, for proper alignment. Don't ask me why it works!:
6314 \bbl@exp{% Avoid a 'visible' conditional

```

```

6315     \\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}\<fi>}}%
6316 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6317 \AddToHook{env/split/before}{%
6318   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6319   \ifnum\bbl@thetextdir>\z@
6320     \bbl@ifsamestring\@currentenv{equation}%
6321     {\ifx\bbl@ams@lap\hbox % leqno
6322       \def\bbl@ams@flip#1{%
6323         \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6324       \else
6325         \def\bbl@ams@flip#1{%
6326           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6327         \fi}%
6328       }%
6329     \fi}%
6330   \fi\fi}
6331 \fi
6332 \def\bbl@provide@extra#1{%
6333   % == Counters: mapdigits ==
6334   % Native digits
6335   \ifx\bbl@KVP@mapdigits\@nnil\else
6336     \bbl@ifunset{\bbl@dgnat@\language\language}\fi%
6337     {\RequirePackage{luatexbase}%
6338      \bbl@activate@preotf
6339      \directlua{
6340        Babel = Babel or {} %% -> presets in luababel
6341        Babel.digits_mapped = true
6342        Babel.digits = Babel.digits or {}
6343        Babel.digits[\the\localeid] =
6344          table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6345        if not Babel.numbers then
6346          function Babel.numbers(head)
6347            local LOCALE = Babel.attr_locale
6348            local GLYPH = node.id'glyph'
6349            local inmath = false
6350            for item in node.traverse(head) do
6351              if not inmath and item.id == GLYPH then
6352                local temp = node.get_attribute(item, LOCALE)
6353                if Babel.digits[temp] then
6354                  local chr = item.char
6355                  if chr > 47 and chr < 58 then
6356                    item.char = Babel.digits[temp][chr-47]
6357                  end
6358                end
6359              elseif item.id == node.id'math' then
6360                inmath = (item.subtype == 0)
6361              end
6362            end
6363            return head
6364          end
6365        end
6366      }%
6367   \fi
6368   % == transforms ==
6369   \ifx\bbl@KVP@transforms\@nnil\else
6370     \def\bbl@elt##1##2##3{%
6371       \in@{transforms.}{##1}%
6372       \ifin@
6373         \def\bbl@tempa{##1}%
6374         \bbl@replace\bbl@tempa{transforms.}{}%
6375         \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6376       \fi}%
6377   \csname bbl@inidata@\language\endcsname

```

```

6378 \bbl@release@transforms\relax % \relax closes the last item.
6379 \fi}
6380 % Start tabular here:
6381 \def\localerestoredirs{%
6382 \ifcase\bbl@thetextdir
6383 \ifnum\textdirection=\z@\else\textdir TLT\fi
6384 \else
6385 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6386 \fi
6387 \ifcase\bbl@thepardir
6388 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6389 \else
6390 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6391 \fi}
6392 \IfBabelLayout{tabular}%
6393 {\chardef\bbl@tabular@mode\tw}% All RTL
6394 {\IfBabelLayout{notabular}%
6395 {\chardef\bbl@tabular@mode\z}%
6396 {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6397 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6398 \ifnum\bbl@tabular@mode=\@ne
6399 \let\bbl@parabefore\relax
6400 \AddToHook{para/before}{\bbl@parabefore}
6401 \AtBeginDocument{%
6402 \bbl@replace\@tabular{$}{%$%
6403 \def\bbl@insidemath{0}%
6404 \def\bbl@parabefore{\localerestoredirs}}%
6405 \ifnum\bbl@tabular@mode=\@ne
6406 \bbl@ifunset{\@tabclassz}{%
6407 \bbl@exp{% Hide conditionals
6408 \\\bbl@sreplace\\\@tabclassz
6409 {\<ifcase>\\\@chnum}%
6410 {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6411 \@ifpackageloaded{colortbl}%
6412 {\bbl@sreplace\@classz
6413 {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6414 {\@ifpackageloaded{array}%
6415 {\bbl@exp{% Hide conditionals
6416 \\\bbl@sreplace\\\@classz
6417 {\<ifcase>\\\@chnum}%
6418 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6419 \\\bbl@sreplace\\\@classz
6420 {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
6421 {}}%
6422 \fi}
6423 \fi
6424 \AtBeginDocument{%
6425 \@ifpackageloaded{multicol}%
6426 {\toks@ \expandafter{\multi@column@out}%
6427 \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6428 {}%
6429 \fi
6430 \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`.

```

6431 \ifnum\bbl@bidimode>\z@ % Any bidi=
6432 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6433 \bbl@exp{%
6434 \def\\bbl@insidemath{0}%
6435 \mathdir\the\bodydir

```

```

6436      #1%                Once entered in math, set boxes to restore values
6437      \<ifmmode>%
6438      \everyvbox{%
6439      \the\everyvbox
6440      \bodydir\the\bodydir
6441      \mathdir\the\mathdir
6442      \everyhbox{\the\everyhbox}%
6443      \everyvbox{\the\everyvbox}}%
6444      \everyhbox{%
6445      \the\everyhbox
6446      \bodydir\the\bodydir
6447      \mathdir\the\mathdir
6448      \everyhbox{\the\everyhbox}%
6449      \everyvbox{\the\everyvbox}}%
6450      \<fi>}}%
6451      \def\@hangfrom#1{%
6452      \setbox\@tempboxa\hbox{#{#1}}%
6453      \hangindent\wd\@tempboxa
6454      \ifnum\bbL@getluadir{page}=\bbL@getluadir{par}\else
6455      \shapemode\@ne
6456      \fi
6457      \noindent\box\@tempboxa}
6458 \fi
6459 \IfBabelLayout{tabular}
6460 {\let\bbL@OL@tabular\@tabular
6461 \bbL@replace\@tabular{$}{\bbL@nextfake$}%
6462 \let\bbL@NL@tabular\@tabular
6463 \AtBeginDocument{%
6464 \ifx\bbL@NL@tabular\@tabular\else
6465 \bbL@replace\@tabular{$}{\bbL@nextfake$}%
6466 \let\bbL@NL@tabular\@tabular
6467 \fi}}
6468 {}
6469 \IfBabelLayout{lists}
6470 {\let\bbL@OL@list\list
6471 \bbL@sreplace\list{\parshape}{\bbL@listparshape}%
6472 \let\bbL@NL@list\list
6473 \def\bbL@listparshape#1#2#3{%
6474 \parshape #1 #2 #3 %
6475 \ifnum\bbL@getluadir{page}=\bbL@getluadir{par}\else
6476 \shapemode\tw@
6477 \fi}}
6478 {}
6479 \IfBabelLayout{graphics}
6480 {\let\bbL@pictresetdir\relax
6481 \def\bbL@pictsetdir#1{%
6482 \ifcase\bbL@thetextdir
6483 \let\bbL@pictresetdir\relax
6484 \else
6485 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6486 \or\textdir TLT
6487 \else\bodydir TLT \textdir TLT
6488 \fi
6489 % \(\text|par)dir required in pgf:
6490 \def\bbL@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6491 \fi}%
6492 \AddToHook{env/picture/begin}{\bbL@pictsetdir\tw@}%
6493 \directlua{
6494 Babel.get_picture_dir = true
6495 Babel.picture_has_bidi = 0
6496 %
6497 function Babel.picture_dir (head)
6498 if not Babel.get_picture_dir then return head end

```

```

6499     if Babel.hlist_has_bidi(head) then
6500         Babel.picture_has_bidi = 1
6501     end
6502     return head
6503 end
6504 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6505     "Babel.picture_dir")
6506 }%
6507 \AtBeginDocument{%
6508     \def\LS@rot{%
6509         \setbox\@outputbox\vbox{%
6510             \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6511     \long\def\put(#1,#2)#3{%
6512         \@killglue
6513         % Try:
6514         \ifx\bbbl@pictresetdir\relax
6515             \def\bbbl@tempc{0}%
6516         \else
6517             \directlua{
6518                 Babel.get_picture_dir = true
6519                 Babel.picture_has_bidi = 0
6520             }%
6521             \setbox\z@\hb@xt@\z@{%
6522                 \@defaultunitsset\@tempdimc{#1}\unitlength
6523                 \kern\@tempdimc
6524                 #3\hss}% TODO0: #3 executed twice (below). That's bad.
6525             \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6526         \fi
6527         % Do:
6528         \@defaultunitsset\@tempdimc{#2}\unitlength
6529         \raise\@tempdimc\hb@xt@\z@{%
6530             \@defaultunitsset\@tempdimc{#1}\unitlength
6531             \kern\@tempdimc
6532             {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6533         \ignorespaces}%
6534     \MakeRobust\put}%
6535 \AtBeginDocument
6536 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir\@gobble}%
6537 \ifx\pgfpicture\@undefined\else % TODO0. Allow deactivate?
6538     \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
6539     \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6540     \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6541 \fi
6542 \ifx\tikzpicture\@undefined\else
6543     \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw@}%
6544     \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6545     \bbbl@sreplace\tikz{\begin@group}{\begin@group\bbbl@pictsetdir\tw@}%
6546 \fi
6547 \ifx\tcolorbox\@undefined\else
6548     \def\tcb@drawing@env@begin{%
6549         \csname tcb@before@tcb@split@state\endcsname
6550         \bbbl@pictsetdir\tw@
6551         \begin{\kv tcb@graphenv}%
6552         \tcb@bbdraw%
6553         \tcb@apply@graph@patches
6554         }%
6555     \def\tcb@drawing@env@end{%
6556         \end{\kv tcb@graphenv}%
6557         \bbbl@pictresetdir
6558         \csname tcb@after@tcb@split@state\endcsname
6559         }%
6560     \fi
6561 }}

```

```
6562 {}
```

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`.

```
6563 \IfBabelLayout{counters*}%
6564 {\bbl@add\bbl@opt@layout{.counters.}%
6565 \directlua{
6566   luatexbase.add_to_callback("process_output_buffer",
6567     Babel.discard_sublr , "Babel.discard_sublr") }%
6568 {}}
6569 \IfBabelLayout{counters}%
6570 {\let\bbl@0L@@textsuperscript\@textsuperscript
6571 \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6572 \let\bbl@latinarabic=\@arabic
6573 \let\bbl@0L@@arabic\@arabic
6574 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6575 \@ifpackagewith{babel}{bidi=default}%
6576 {\let\bbl@asciroman=\@roman
6577 \let\bbl@0L@@roman\@roman
6578 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6579 \let\bbl@asciRoman=\@Roman
6580 \let\bbl@0L@@roman\@Roman
6581 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciRoman#1}}}%
6582 \let\bbl@0L@labelenumii\labelenumii
6583 \def\labelenumii{}\theenumii}%
6584 \let\bbl@0L@p@enumiii\p@enumiii
6585 \def\p@enumiii{\p@enumii}\theenumii{}\}\}\}\}
6586 <<Footnote changes>>
6587 \IfBabelLayout{footnotes}%
6588 {\let\bbl@0L@footnote\footnote
6589 \BabelFootnote\footnote\language\language{}{}%
6590 \BabelFootnote\localfootnote\language\language{}{}%
6591 \BabelFootnote\mainfootnote{}{}{}%
6592 {}
```

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.

```
6593 \IfBabelLayout{extras}%
6594 {\let\bbl@0L@underline\underline
6595 \bbl@sreplace\underline{\$@@underline}{\bbl@nextfake$@@underline}%
6596 \let\bbl@0L@LaTeX2e\LaTeX2e
6597 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6598 \if b\expandafter\@car\@series\@nil\boldmath\fi
6599 \babelsublr{%
6600 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6601 {}
6602 </luatex>
```

9.12 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.

```
6603 <{*transforms>
```

```

6604 Babel.linebreaking.replacements = {}
6605 Babel.linebreaking.replacements[0] = {} -- pre
6606 Babel.linebreaking.replacements[1] = {} -- post
6607 Babel.linebreaking.replacements[2] = {} -- post-line WIP
6608
6609 -- Discretionaries contain strings as nodes
6610 function Babel.str_to_nodes(fn, matches, base)
6611   local n, head, last
6612   if fn == nil then return nil end
6613   for s in string.utfvalues(fn(matches)) do
6614     if base.id == 7 then
6615       base = base.replace
6616     end
6617     n = node.copy(base)
6618     n.char = s
6619     if not head then
6620       head = n
6621     else
6622       last.next = n
6623     end
6624     last = n
6625   end
6626   return head
6627 end
6628
6629 Babel.fetch_subtext = {}
6630
6631 Babel.ignore_pre_char = function(node)
6632   return (node.lang == Babel.nohyphenation)
6633 end
6634
6635 -- Merging both functions doesn't seem feasible, because there are too
6636 -- many differences.
6637 Babel.fetch_subtext[0] = function(head)
6638   local word_string = ''
6639   local word_nodes = {}
6640   local lang
6641   local item = head
6642   local inmath = false
6643
6644   while item do
6645     if item.id == 11 then
6646       inmath = (item.subtype == 0)
6647     end
6648
6649     if inmath then
6650       -- pass
6651     elseif item.id == 29 then
6652       local locale = node.get_attribute(item, Babel.attr_locale)
6653
6654       if lang == locale or lang == nil then
6655         lang = lang or locale
6656         if Babel.ignore_pre_char(item) then
6657           word_string = word_string .. Babel.us_char
6658         else
6659           word_string = word_string .. unicode.utf8.char(item.char)
6660         end
6661       end
6662       word_nodes[#word_nodes+1] = item
6663     else
6664       break
6665     end
6666   end

```

```

6667
6668     elseif item.id == 12 and item.subtype == 13 then
6669         word_string = word_string .. ' '
6670         word_nodes[#word_nodes+1] = item
6671
6672         -- Ignore leading unrecognized nodes, too.
6673     elseif word_string ~= '' then
6674         word_string = word_string .. Babel.us_char
6675         word_nodes[#word_nodes+1] = item -- Will be ignored
6676     end
6677
6678     item = item.next
6679 end
6680
6681 -- Here and above we remove some trailing chars but not the
6682 -- corresponding nodes. But they aren't accessed.
6683 if word_string:sub(-1) == ' ' then
6684     word_string = word_string:sub(1,-2)
6685 end
6686 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6687 return word_string, word_nodes, item, lang
6688 end
6689
6690 Babel.fetch_subtext[1] = function(head)
6691     local word_string = ''
6692     local word_nodes = {}
6693     local lang
6694     local item = head
6695     local inmath = false
6696
6697     while item do
6698
6699         if item.id == 11 then
6700             inmath = (item.subtype == 0)
6701         end
6702
6703         if inmath then
6704             -- pass
6705
6706         elseif item.id == 29 then
6707             if item.lang == lang or lang == nil then
6708                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6709                     lang = lang or item.lang
6710                     word_string = word_string .. unicode.utf8.char(item.char)
6711                     word_nodes[#word_nodes+1] = item
6712                 end
6713             else
6714                 break
6715             end
6716
6717         elseif item.id == 7 and item.subtype == 2 then
6718             word_string = word_string .. '='
6719             word_nodes[#word_nodes+1] = item
6720
6721         elseif item.id == 7 and item.subtype == 3 then
6722             word_string = word_string .. '|'
6723             word_nodes[#word_nodes+1] = item
6724
6725         -- (1) Go to next word if nothing was found, and (2) implicitly
6726         -- remove leading USs.
6727         elseif word_string == '' then
6728             -- pass
6729

```



```

6730 -- This is the responsible for splitting by words.
6731 elseif (item.id == 12 and item.subtype == 13) then
6732     break
6733
6734 else
6735     word_string = word_string .. Babel.us_char
6736     word_nodes[#word_nodes+1] = item -- Will be ignored
6737 end
6738
6739 item = item.next
6740 end
6741
6742 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6743 return word_string, word_nodes, item, lang
6744 end
6745
6746 function Babel.pre_hyphenate_replace(head)
6747     Babel.hyphenate_replace(head, 0)
6748 end
6749
6750 function Babel.post_hyphenate_replace(head)
6751     Babel.hyphenate_replace(head, 1)
6752 end
6753
6754 Babel.us_char = string.char(31)
6755
6756 function Babel.hyphenate_replace(head, mode)
6757     local u = unicode.utf8
6758     local lbkr = Babel.linebreaking.replacements[mode]
6759     if mode == 2 then mode = 0 end -- WIP
6760
6761     local word_head = head
6762
6763     while true do -- for each subtext block
6764
6765         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6766
6767         if Babel.debug then
6768             print()
6769             print((mode == 0) and '@@@@<' or '@@@@>', w)
6770         end
6771
6772         if nw == nil and w == '' then break end
6773
6774         if not lang then goto next end
6775         if not lbkr[lang] then goto next end
6776
6777         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6778         -- loops are nested.
6779         for k=1, #lbkr[lang] do
6780             local p = lbkr[lang][k].pattern
6781             local r = lbkr[lang][k].replace
6782             local attr = lbkr[lang][k].attr or -1
6783
6784             if Babel.debug then
6785                 print('*****', p, mode)
6786             end
6787
6788             -- This variable is set in some cases below to the first *byte*
6789             -- after the match, either as found by u.match (faster) or the
6790             -- computed position based on sc if w has changed.
6791             local last_match = 0
6792             local step = 0

```

```

6793
6794 -- For every match.
6795 while true do
6796     if Babel.debug then
6797         print('====')
6798     end
6799     local new -- used when inserting and removing nodes
6800
6801     local matches = { u.match(w, p, last_match) }
6802
6803     if #matches < 2 then break end
6804
6805     -- Get and remove empty captures (with ())'s, which return a
6806     -- number with the position), and keep actual captures
6807     -- (from (...)), if any, in matches.
6808     local first = table.remove(matches, 1)
6809     local last = table.remove(matches, #matches)
6810     -- Non re-fetched substrings may contain \31, which separates
6811     -- subsubstrings.
6812     if string.find(w:sub(first, last-1), Babel.us_char) then break end
6813
6814     local save_last = last -- with A()BC()D, points to D
6815
6816     -- Fix offsets, from bytes to unicode. Explained above.
6817     first = u.len(w:sub(1, first-1)) + 1
6818     last = u.len(w:sub(1, last-1)) -- now last points to C
6819
6820     -- This loop stores in a small table the nodes
6821     -- corresponding to the pattern. Used by 'data' to provide a
6822     -- predictable behavior with 'insert' (w_nodes is modified on
6823     -- the fly), and also access to 'remove'd nodes.
6824     local sc = first-1 -- Used below, too
6825     local data_nodes = {}
6826
6827     local enabled = true
6828     for q = 1, last-first+1 do
6829         data_nodes[q] = w_nodes[sc+q]
6830         if enabled
6831             and attr > -1
6832             and not node.has_attribute(data_nodes[q], attr)
6833         then
6834             enabled = false
6835         end
6836     end
6837
6838     -- This loop traverses the matched substring and takes the
6839     -- corresponding action stored in the replacement list.
6840     -- sc = the position in substr nodes / string
6841     -- rc = the replacement table index
6842     local rc = 0
6843
6844     while rc < last-first+1 do -- for each replacement
6845         if Babel.debug then
6846             print('.....', rc + 1)
6847         end
6848         sc = sc + 1
6849         rc = rc + 1
6850
6851         if Babel.debug then
6852             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6853             local ss = ''
6854             for itt in node.traverse(head) do
6855                 if itt.id == 29 then

```

```

6856         ss = ss .. unicode.utf8.char(itt.char)
6857     else
6858         ss = ss .. '{' .. itt.id .. '}'
6859     end
6860 end
6861 print('*****', ss)
6862
6863 end
6864
6865 local crep = r[rc]
6866 local item = w_nodes[sc]
6867 local item_base = item
6868 local placeholder = Babel.us_char
6869 local d
6870
6871 if crep and crep.data then
6872     item_base = data_nodes[crep.data]
6873 end
6874
6875 if crep then
6876     step = crep.step or 0
6877 end
6878
6879 if (not enabled) or (crep and next(crep) == nil) then -- = {}
6880     last_match = save_last    -- Optimization
6881     goto next
6882
6883 elseif crep == nil or crep.remove then
6884     node.remove(head, item)
6885     table.remove(w_nodes, sc)
6886     w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6887     sc = sc - 1 -- Nothing has been inserted.
6888     last_match = utf8.offset(w, sc+1+step)
6889     goto next
6890
6891 elseif crep and crep.kashida then -- Experimental
6892     node.set_attribute(item,
6893         Babel.attr_kashida,
6894         crep.kashida)
6895     last_match = utf8.offset(w, sc+1+step)
6896     goto next
6897
6898 elseif crep and crep.string then
6899     local str = crep.string(matches)
6900     if str == '' then -- Gather with nil
6901         node.remove(head, item)
6902         table.remove(w_nodes, sc)
6903         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6904         sc = sc - 1 -- Nothing has been inserted.
6905     else
6906         local loop_first = true
6907         for s in string.utfvalues(str) do
6908             d = node.copy(item_base)
6909             d.char = s
6910             if loop_first then
6911                 loop_first = false
6912                 head, new = node.insert_before(head, item, d)
6913                 if sc == 1 then
6914                     word_head = head
6915                 end
6916                 w_nodes[sc] = d
6917                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6918             else

```

```

6919         sc = sc + 1
6920         head, new = node.insert_before(head, item, d)
6921         table.insert(w_nodes, sc, new)
6922         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6923     end
6924     if Babel.debug then
6925         print('.....', 'str')
6926         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6927     end
6928     end -- for
6929     node.remove(head, item)
6930 end -- if ''
6931 last_match = utf8.offset(w, sc+1+step)
6932 goto next
6933
6934 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6935     d = node.new(7, 3) -- (disc, regular)
6936     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6937     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6938     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6939     d.attr = item_base.attr
6940     if crep.pre == nil then -- TeXbook p96
6941         d.penalty = crep.penalty or tex.hyphenpenalty
6942     else
6943         d.penalty = crep.penalty or tex.exhyphenpenalty
6944     end
6945     placeholder = '|'
6946     head, new = node.insert_before(head, item, d)
6947
6948 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6949     -- ERROR
6950
6951 elseif crep and crep.penalty then
6952     d = node.new(14, 0) -- (penalty, userpenalty)
6953     d.attr = item_base.attr
6954     d.penalty = crep.penalty
6955     head, new = node.insert_before(head, item, d)
6956
6957 elseif crep and crep.space then
6958     -- 655360 = 10 pt = 10 * 65536 sp
6959     d = node.new(12, 13) -- (glue, spaceskip)
6960     local quad = font.getfont(item_base.font).size or 655360
6961     node.setglue(d, crep.space[1] * quad,
6962                  crep.space[2] * quad,
6963                  crep.space[3] * quad)
6964     if mode == 0 then
6965         placeholder = ' '
6966     end
6967     head, new = node.insert_before(head, item, d)
6968
6969 elseif crep and crep.spacefactor then
6970     d = node.new(12, 13) -- (glue, spaceskip)
6971     local base_font = font.getfont(item_base.font)
6972     node.setglue(d,
6973                  crep.spacefactor[1] * base_font.parameters['space'],
6974                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6975                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6976     if mode == 0 then
6977         placeholder = ' '
6978     end
6979     head, new = node.insert_before(head, item, d)
6980
6981 elseif mode == 0 and crep and crep.space then

```

```

6982         -- ERROR
6983
6984     end -- ie replacement cases
6985
6986     -- Shared by disc, space and penalty.
6987     if sc == 1 then
6988         word_head = head
6989     end
6990     if crep.insert then
6991         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6992         table.insert(w_nodes, sc, new)
6993         last = last + 1
6994     else
6995         w_nodes[sc] = d
6996         node.remove(head, item)
6997         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6998     end
6999
7000     last_match = utf8.offset(w, sc+1+step)
7001
7002     ::next::
7003
7004     end -- for each replacement
7005
7006     if Babel.debug then
7007         print('.....', '/')
7008         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7009     end
7010
7011     end -- for match
7012
7013     end -- for patterns
7014
7015     ::next::
7016     word_head = nw
7017     end -- for substring
7018     return head
7019 end
7020
7021 -- This table stores capture maps, numbered consecutively
7022 Babel.capture_maps = {}
7023
7024 -- The following functions belong to the next macro
7025 function Babel.capture_func(key, cap)
7026     local ret = "[" .. cap:gsub('{{([0-9])}}', "]]..m[%1]..[" .. "]"
7027     local cnt
7028     local u = unicode.utf8
7029     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(.-)}}', Babel.capture_func_map)
7030     if cnt == 0 then
7031         ret = u.gsub(ret, '{(%x%x%x%x+)}',
7032             function (n)
7033                 return u.char(tonumber(n, 16))
7034             end)
7035     end
7036     ret = ret:gsub("%[%[%]]%.", '')
7037     ret = ret:gsub("%.%[%[%]]%", '')
7038     return key .. "[=function(m) return ]] .. ret .. [[ end]]
7039 end
7040
7041 function Babel.capt_map(from, mapno)
7042     return Babel.capture_maps[mapno][from] or from
7043 end
7044

```

```

7045 -- Handle the {n|abc|ABC} syntax in captures
7046 function Babel.capture_func_map(capno, from, to)
7047     local u = unicode.utf8
7048     from = u.gsub(from, '{(%x%x%x%x+)}',
7049         function (n)
7050             return u.char(tonumber(n, 16))
7051         end)
7052     to = u.gsub(to, '{(%x%x%x%x+)}',
7053         function (n)
7054             return u.char(tonumber(n, 16))
7055         end)
7056     local froms = {}
7057     for s in string.utfcharacters(from) do
7058         table.insert(froms, s)
7059     end
7060     local cnt = 1
7061     table.insert(Babel.capture_maps, {})
7062     local mlen = table.getn(Babel.capture_maps)
7063     for s in string.utfcharacters(to) do
7064         Babel.capture_maps[mlen][froms[cnt]] = s
7065         cnt = cnt + 1
7066     end
7067     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7068         (mlen) .. ").." .. "["
7069 end
7070
7071 -- Create/Extend reversed sorted list of kashida weights:
7072 function Babel.capture_kashida(key, wt)
7073     wt = tonumber(wt)
7074     if Babel.kashida_wts then
7075         for p, q in ipairs(Babel.kashida_wts) do
7076             if wt == q then
7077                 break
7078             elseif wt > q then
7079                 table.insert(Babel.kashida_wts, p, wt)
7080                 break
7081             elseif table.getn(Babel.kashida_wts) == p then
7082                 table.insert(Babel.kashida_wts, wt)
7083             end
7084         end
7085     else
7086         Babel.kashida_wts = { wt }
7087     end
7088     return 'kashida = ' .. wt
7089 end
7090 </transforms>

```

9.13 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.

```

7091 (*basic-r)
7092 Babel = Babel or {}
7093
7094 Babel.bidi_enabled = true
7095
7096 require('babel-data-bidi.lua')
7097
7098 local characters = Babel.characters
7099 local ranges = Babel.ranges
7100
7101 local DIR = node.id("dir")
7102
7103 local function dir_mark(head, from, to, outer)
7104   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7105   local d = node.new(DIR)
7106   d.dir = '+' .. dir
7107   node.insert_before(head, from, d)
7108   d = node.new(DIR)
7109   d.dir = '-' .. dir
7110   node.insert_after(head, to, d)
7111 end
7112
7113 function Babel.bidi(head, ispar)
7114   local first_n, last_n          -- first and last char with nums
7115   local last_es                  -- an auxiliary 'last' used with nums
7116   local first_d, last_d          -- first and last char in L/R block
7117   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):

```

7118   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7119   local strong_lr = (strong == 'l') and 'l' or 'r'
7120   local outer = strong
7121
7122   local new_dir = false
7123   local first_dir = false
7124   local inmath = false
7125
7126   local last_lr
7127

```

```

7128 local type_n = ''
7129
7130 for item in node.traverse(head) do
7131     -- three cases: glyph, dir, otherwise
7132     if item.id == node.id'glyph'
7133         or (item.id == 7 and item.subtype == 2) then
7134
7135         local itemchar
7136         if item.id == 7 and item.subtype == 2 then
7137             itemchar = item.replace.char
7138         else
7139             itemchar = item.char
7140         end
7141         local chardata = characters[itemchar]
7142         dir = chardata and chardata.d or nil
7143         if not dir then
7144             for nn, et in ipairs(ranges) do
7145                 if itemchar < et[1] then
7146                     break
7147                 elseif itemchar <= et[2] then
7148                     dir = et[3]
7149                     break
7150                 end
7151             end
7152         end
7153         dir = dir or 'l'
7154         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).

```

7156 if new_dir then
7157     attr_dir = 0
7158     for at in node.traverse(item.attr) do
7159         if at.number == Babel.attr_dir then
7160             attr_dir = at.value & 0x3
7161         end
7162     end
7163     if attr_dir == 1 then
7164         strong = 'r'
7165     elseif attr_dir == 2 then
7166         strong = 'al'
7167     else
7168         strong = 'l'
7169     end
7170     strong_lr = (strong == 'l') and 'l' or 'r'
7171     outer = strong_lr
7172     new_dir = false
7173 end
7174
7175 if dir == 'nsm' then dir = strong end -- W1

```

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

```

7176 dir_real = dir -- We need dir_real to set strong below
7177 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:

```

7178 if strong == 'al' then
7179     if dir == 'en' then dir = 'an' end -- W2
7180     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6

```



```

7181         strong_lr = 'r'                                -- W3
7182     end

```

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

```

7183     elseif item.id == node.id'dir' and not inmath then
7184         new_dir = true
7185         dir = nil
7186     elseif item.id == node.id'math' then
7187         inmath = (item.subtype == 0)
7188     else
7189         dir = nil          -- Not a char
7190     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>.

```

7191     if dir == 'en' or dir == 'an' or dir == 'et' then
7192         if dir ~= 'et' then
7193             type_n = dir
7194         end
7195         first_n = first_n or item
7196         last_n = last_n or item
7197         last_es = nil
7198     elseif dir == 'es' and last_n then -- W3+W6
7199         last_es = item
7200     elseif dir == 'cs' then          -- it's right - do nothing
7201     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7202         if strong_lr == 'r' and type_n ~= '' then
7203             dir_mark(head, first_n, last_n, 'r')
7204         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7205             dir_mark(head, first_n, last_n, 'r')
7206             dir_mark(head, first_d, last_d, outer)
7207             first_d, last_d = nil, nil
7208         elseif strong_lr == 'l' and type_n ~= '' then
7209             last_d = last_n
7210         end
7211         type_n = ''
7212         first_n, last_n = nil, nil
7213     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:

```

7214     if dir == 'l' or dir == 'r' then
7215         if dir ~= outer then
7216             first_d = first_d or item
7217             last_d = item
7218         elseif first_d and dir ~= strong_lr then
7219             dir_mark(head, first_d, last_d, outer)
7220             first_d, last_d = nil, nil
7221         end
7222     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.

```

7223     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7224         item.char = characters[item.char] and
7225             characters[item.char].m or item.char

```

```

7226 elseif (dir or new_dir) and last_lr ~= item then
7227     local mir = outer .. strong_lr .. (dir or outer)
7228     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7229         for ch in node.traverse(node.next(last_lr)) do
7230             if ch == item then break end
7231             if ch.id == node.id'glyph' and characters[ch.char] then
7232                 ch.char = characters[ch.char].m or ch.char
7233             end
7234         end
7235     end
7236 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).

```

7237 if dir == 'l' or dir == 'r' then
7238     last_lr = item
7239     strong = dir_real -- Don't search back - best save now
7240     strong_lr = (strong == 'l') and 'l' or 'r'
7241 elseif new_dir then
7242     last_lr = nil
7243 end
7244 end

```

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

```

7245 if last_lr and outer == 'r' then
7246     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7247         if characters[ch.char] then
7248             ch.char = characters[ch.char].m or ch.char
7249         end
7250     end
7251 end
7252 if first_n then
7253     dir_mark(head, first_n, last_n, outer)
7254 end
7255 if first_d then
7256     dir_mark(head, first_d, last_d, outer)
7257 end

```

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

```

7258 return node.prev(head) or head
7259 end
7260 </basic-r>

```

And here the Lua code for bidi=basic:

```

7261 (*basic)
7262 Babel = Babel or {}
7263
7264 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7265
7266 Babel.fontmap = Babel.fontmap or {}
7267 Babel.fontmap[0] = {} -- l
7268 Babel.fontmap[1] = {} -- r
7269 Babel.fontmap[2] = {} -- al/an
7270
7271 Babel.bidi_enabled = true
7272 Babel.mirroring_enabled = true
7273
7274 require('babel-data-bidi.lua')
7275
7276 local characters = Babel.characters
7277 local ranges = Babel.ranges
7278
7279 local DIR = node.id('dir')

```

```

7280 local GLYPH = node.id('glyph')
7281
7282 local function insert_implicit(head, state, outer)
7283   local new_state = state
7284   if state.sim and state.eim and state.sim ~= state.eim then
7285     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7286     local d = node.new(DIR)
7287     d.dir = '+' .. dir
7288     node.insert_before(head, state.sim, d)
7289     local d = node.new(DIR)
7290     d.dir = '-' .. dir
7291     node.insert_after(head, state.eim, d)
7292   end
7293   new_state.sim, new_state.eim = nil, nil
7294   return head, new_state
7295 end
7296
7297 local function insert_numeric(head, state)
7298   local new
7299   local new_state = state
7300   if state.san and state.ean and state.san ~= state.ean then
7301     local d = node.new(DIR)
7302     d.dir = '+TLT'
7303     _, new = node.insert_before(head, state.san, d)
7304     if state.san == state.sim then state.sim = new end
7305     local d = node.new(DIR)
7306     d.dir = '-TLT'
7307     _, new = node.insert_after(head, state.ean, d)
7308     if state.ean == state.eim then state.eim = new end
7309   end
7310   new_state.san, new_state.ean = nil, nil
7311   return head, new_state
7312 end
7313
7314 -- TODO - \hbox with an explicit dir can lead to wrong results
7315 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7316 -- was s made to improve the situation, but the problem is the 3-dir
7317 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7318 -- well.
7319
7320 function Babel.bidi(head, ispar, hdir)
7321   local d -- d is used mainly for computations in a loop
7322   local prev_d = ''
7323   local new_d = false
7324
7325   local nodes = {}
7326   local outer_first = nil
7327   local inmath = false
7328
7329   local glue_d = nil
7330   local glue_i = nil
7331
7332   local has_en = false
7333   local first_et = nil
7334
7335   local has_hyperlink = false
7336
7337   local ATDIR = Babel.attr_dir
7338
7339   local save_outer
7340   local temp = node.get_attribute(head, ATDIR)
7341   if temp then
7342     temp = temp & 0x3

```

```

7343     save_outter = (temp == 0 and 'l') or
7344                   (temp == 1 and 'r') or
7345                   (temp == 2 and 'al')
7346 elseif ispar then      -- Or error? Shouldn't happen
7347     save_outter = ('TRT' == tex.pardir) and 'r' or 'l'
7348 else                  -- Or error? Shouldn't happen
7349     save_outter = ('TRT' == hdir) and 'r' or 'l'
7350 end
7351 -- when the callback is called, we are just _after_ the box,
7352 -- and the textdir is that of the surrounding text
7353 -- if not ispar and hdir ~= tex.textdir then
7354 --     save_outter = ('TRT' == hdir) and 'r' or 'l'
7355 -- end
7356 local outer = save_outter
7357 local last = outer
7358 -- 'al' is only taken into account in the first, current loop
7359 if save_outter == 'al' then save_outter = 'r' end
7360
7361 local fontmap = Babel.fontmap
7362
7363 for item in node.traverse(head) do
7364
7365     -- In what follows, #node is the last (previous) node, because the
7366     -- current one is not added until we start processing the neutrals.
7367
7368     -- three cases: glyph, dir, otherwise
7369     if item.id == GLYPH
7370     or (item.id == 7 and item.subtype == 2) then
7371
7372         local d_font = nil
7373         local item_r
7374         if item.id == 7 and item.subtype == 2 then
7375             item_r = item.replace    -- automatic discs have just 1 glyph
7376         else
7377             item_r = item
7378         end
7379         local chardata = characters[item_r.char]
7380         d = chardata and chardata.d or nil
7381         if not d or d == 'nsm' then
7382             for nn, et in ipairs(ranges) do
7383                 if item_r.char < et[1] then
7384                     break
7385                 elseif item_r.char <= et[2] then
7386                     if not d then d = et[3]
7387                     elseif d == 'nsm' then d_font = et[3]
7388                     end
7389                     break
7390             end
7391         end
7392         d = d or 'l'
7393
7394         -- A short 'pause' in bidi for mapfont
7395         d_font = d_font or d
7396         d_font = (d_font == 'l' and 0) or
7397                 (d_font == 'nsm' and 0) or
7398                 (d_font == 'r' and 1) or
7399                 (d_font == 'al' and 2) or
7400                 (d_font == 'an' and 2) or nil
7401         if d_font and fontmap and fontmap[d_font][item_r.font] then
7402             item_r.font = fontmap[d_font][item_r.font]
7403         end
7404     end
7405 end

```

```

7406     if new_d then
7407         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7408     if inmath then
7409         attr_d = 0
7410     else
7411         attr_d = node.get_attribute(item, ATDIR)
7412         attr_d = attr_d & 0x3
7413     end
7414     if attr_d == 1 then
7415         outer_first = 'r'
7416         last = 'r'
7417     elseif attr_d == 2 then
7418         outer_first = 'r'
7419         last = 'al'
7420     else
7421         outer_first = 'l'
7422         last = 'l'
7423     end
7424     outer = last
7425     has_en = false
7426     first_et = nil
7427     new_d = false
7428 end
7429
7430 if glue_d then
7431     if (d == 'l' and 'l' or 'r') ~= glue_d then
7432         table.insert(nodes, {glue_i, 'on', nil})
7433     end
7434     glue_d = nil
7435     glue_i = nil
7436 end
7437
7438 elseif item.id == DIR then
7439     d = nil
7440
7441     if head ~= item then new_d = true end
7442
7443 elseif item.id == node.id'glue' and item.subtype == 13 then
7444     glue_d = d
7445     glue_i = item
7446     d = nil
7447
7448 elseif item.id == node.id'math' then
7449     inmath = (item.subtype == 0)
7450
7451 elseif item.id == 8 and item.subtype == 19 then
7452     has_hyperlink = true
7453
7454 else
7455     d = nil
7456 end
7457
7458 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7459 if last == 'al' and d == 'en' then
7460     d = 'an'           -- W3
7461 elseif last == 'al' and (d == 'et' or d == 'es') then
7462     d = 'on'           -- W6
7463 end
7464
7465 -- EN + CS/ES + EN      -- W4
7466 if d == 'en' and #nodes >= 2 then
7467     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7468         and nodes[#nodes-1][2] == 'en' then

```

```

7469         nodes[#nodes][2] = 'en'
7470     end
7471 end
7472
7473 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7474 if d == 'an' and #nodes >= 2 then
7475     if (nodes[#nodes][2] == 'cs')
7476         and nodes[#nodes-1][2] == 'an' then
7477         nodes[#nodes][2] = 'an'
7478     end
7479 end
7480
7481 -- ET/EN                  -- W5 + W7->l / W6->on
7482 if d == 'et' then
7483     first_et = first_et or (#nodes + 1)
7484 elseif d == 'en' then
7485     has_en = true
7486     first_et = first_et or (#nodes + 1)
7487 elseif first_et then      -- d may be nil here !
7488     if has_en then
7489         if last == 'l' then
7490             temp = 'l'    -- W7
7491         else
7492             temp = 'en'   -- W5
7493         end
7494     else
7495         temp = 'on'      -- W6
7496     end
7497     for e = first_et, #nodes do
7498         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7499     end
7500     first_et = nil
7501     has_en = false
7502 end
7503
7504 -- Force mathdir in math if ON (currently works as expected only
7505 -- with 'l')
7506 if inmath and d == 'on' then
7507     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7508 end
7509
7510 if d then
7511     if d == 'al' then
7512         d = 'r'
7513         last = 'al'
7514     elseif d == 'l' or d == 'r' then
7515         last = d
7516     end
7517     prev_d = d
7518     table.insert(nodes, {item, d, outer_first})
7519 end
7520
7521 outer_first = nil
7522
7523 end
7524
7525 -- TODO -- repeated here in case EN/ET is the last node. Find a
7526 -- better way of doing things:
7527 if first_et then      -- dir may be nil here !
7528     if has_en then
7529         if last == 'l' then
7530             temp = 'l'    -- W7
7531         else

```

```

7532         temp = 'en'    -- W5
7533     end
7534 else
7535     temp = 'on'        -- W6
7536 end
7537 for e = first_et, #nodes do
7538     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7539 end
7540 end
7541
7542 -- dummy node, to close things
7543 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7544
7545 ----- NEUTRAL -----
7546
7547 outer = save_outer
7548 last = outer
7549
7550 local first_on = nil
7551
7552 for q = 1, #nodes do
7553     local item
7554
7555     local outer_first = nodes[q][3]
7556     outer = outer_first or outer
7557     last = outer_first or last
7558
7559     local d = nodes[q][2]
7560     if d == 'an' or d == 'en' then d = 'r' end
7561     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7562
7563     if d == 'on' then
7564         first_on = first_on or q
7565     elseif first_on then
7566         if last == d then
7567             temp = d
7568         else
7569             temp = outer
7570         end
7571         for r = first_on, q - 1 do
7572             nodes[r][2] = temp
7573             item = nodes[r][1]    -- MIRRORING
7574             if Babel.mirroring_enabled and item.id == GLYPH
7575                 and temp == 'r' and characters[item.char] then
7576                 local font_mode = ''
7577                 if item.font > 0 and font.fonts[item.font].properties then
7578                     font_mode = font.fonts[item.font].properties.mode
7579                 end
7580                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7581                     item.char = characters[item.char].m or item.char
7582                 end
7583             end
7584         end
7585         first_on = nil
7586     end
7587
7588     if d == 'r' or d == 'l' then last = d end
7589 end
7590
7591 ----- IMPLICIT, REORDER -----
7592
7593 outer = save_outer
7594 last = outer

```

```

7595
7596 local state = {}
7597 state.has_r = false
7598
7599 for q = 1, #nodes do
7600     local item = nodes[q][1]
7601
7602     outer = nodes[q][3] or outer
7603
7604     local d = nodes[q][2]
7605
7606     if d == 'nsm' then d = last end          -- W1
7607     if d == 'en' then d = 'an' end
7608     local isdir = (d == 'r' or d == 'l')
7609
7610     if outer == 'l' and d == 'an' then
7611         state.san = state.san or item
7612         state.ean = item
7613     elseif state.san then
7614         head, state = insert_numeric(head, state)
7615     end
7616
7617     if outer == 'l' then
7618         if d == 'an' or d == 'r' then      -- im -> implicit
7619             if d == 'r' then state.has_r = true end
7620             state.sim = state.sim or item
7621             state.eim = item
7622         elseif d == 'l' and state.sim and state.has_r then
7623             head, state = insert_implicit(head, state, outer)
7624         elseif d == 'l' then
7625             state.sim, state.eim, state.has_r = nil, nil, false
7626         end
7627     else
7628         if d == 'an' or d == 'l' then
7629             if nodes[q][3] then -- nil except after an explicit dir
7630                 state.sim = item -- so we move sim 'inside' the group
7631             else
7632                 state.sim = state.sim or item
7633             end
7634             state.eim = item
7635         elseif d == 'r' and state.sim then
7636             head, state = insert_implicit(head, state, outer)
7637         elseif d == 'r' then
7638             state.sim, state.eim = nil, nil
7639         end
7640     end
7641 end
7642
7643 if isdir then
7644     last = d          -- Don't search back - best save now
7645 elseif d == 'on' and state.san then
7646     state.san = state.san or item
7647     state.ean = item
7648 end
7649
7650 end
7651
7652 head = node.prev(head) or head
7653
7654 ----- FIX HYPERLINKS -----
7655
7656 if has_hyperlink then
7657     local flag, linking = 0, 0

```



```

7658   for item in node.traverse(head) do
7659     if item.id == DIR then
7660       if item.dir == '+TRT' or item.dir == '+TLT' then
7661         flag = flag + 1
7662       elseif item.dir == '-TRT' or item.dir == '-TLT' then
7663         flag = flag - 1
7664       end
7665     elseif item.id == 8 and item.subtype == 19 then
7666       linking = flag
7667     elseif item.id == 8 and item.subtype == 20 then
7668       if linking > 0 then
7669         if item.prev.id == DIR and
7670           (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7671           d = node.new(DIR)
7672           d.dir = item.prev.dir
7673           node.remove(head, item.prev)
7674           node.insert_after(head, item, d)
7675         end
7676       end
7677       linking = 0
7678     end
7679   end
7680 end
7681
7682 return head
7683 end
7684 </basic>

```

10 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.

11 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.

```

7685 <nil>
7686 \ProvidesLanguage{nil}[\<date>] v\<version> Nil language]
7687 \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.

```

7688 \ifx\l@nil\undefined
7689   \newlanguage\l@nil
7690   \@namedef{bbl@hyphendata@the\l@nil}{{}}}% Remove warning
7691   \let\bbl@elt\relax
7692   \edef\bbl@languages{% Add it to the list of languages
7693     \bbl@languages\bbl@elt{nil}{the\l@nil}{{}}
7694 \fi

```

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

```
7695 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
```

```
\datenil
```

```
7696 \let\captionsnil\@empty
```

```
7697 \let\datenil\@empty
```

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

```
7698 \def\bbl@inidata@nil{%
7699   \bbl@elt{identification}{tag.ini}{und}%
7700   \bbl@elt{identification}{load.level}{0}%
7701   \bbl@elt{identification}{charset}{utf8}%
7702   \bbl@elt{identification}{version}{1.0}%
7703   \bbl@elt{identification}{date}{2022-05-16}%
7704   \bbl@elt{identification}{name.local}{nil}%
7705   \bbl@elt{identification}{name.english}{nil}%
7706   \bbl@elt{identification}{name.babel}{nil}%
7707   \bbl@elt{identification}{tag.bcp47}{und}%
7708   \bbl@elt{identification}{language.tag.bcp47}{und}%
7709   \bbl@elt{identification}{tag.opentype}{dflt}%
7710   \bbl@elt{identification}{script.name}{Latin}%
7711   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7712   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7713   \bbl@elt{identification}{level}{1}%
7714   \bbl@elt{identification}{encodings}{}%
7715   \bbl@elt{identification}{derivate}{no}}
7716 \@namedef{bbl@tbcp@nil}{und}
7717 \@namedef{bbl@lbcp@nil}{und}
7718 \@namedef{bbl@casing@nil}{und} % TODO
7719 \@namedef{bbl@lotf@nil}{dflt}
7720 \@namedef{bbl@elname@nil}{nil}
7721 \@namedef{bbl@lname@nil}{nil}
7722 \@namedef{bbl@esname@nil}{Latin}
7723 \@namedef{bbl@sname@nil}{Latin}
7724 \@namedef{bbl@sbc47@nil}{Latn}
7725 \@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.

```
7726 \ldf@finish{nil}
```

```
7727 \</nil>
```

12 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.

```
7728 <<*Compute Julian day>> ≡
7729 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
7730 \def\bbl@cs@gregleap#1{%
7731   (\bbl@fpmmod{#1}{4} == 0) &&
7732   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
7733 \def\bbl@cs@jd#1#2#3{% year, month, day
7734   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7735     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7736     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
7737     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
7738 <</Compute Julian day>>
```

12.1 Islamic

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

```
7739 (*ca-islamic)
7740 \ExplSyntaxOn
7741 <<Compute Julian day>>
7742 % == islamic (default)
7743 % Not yet implemented
7744 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}
```

The Civil calendar:

```
7745 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7746   ((#3 + ceil(29.5 * (#2 - 1)) +
7747     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7748     1948439.5) - 1) }
7749 \namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl@x{+2}}
7750 \namedef\bbl@ca@islamic-civil+{\bbl@ca@islamicvl@x{+1}}
7751 \namedef\bbl@ca@islamic-civil{\bbl@ca@islamicvl@x{}}
7752 \namedef\bbl@ca@islamic-civil-{\bbl@ca@islamicvl@x{-1}}
7753 \namedef\bbl@ca@islamic-civil--{\bbl@ca@islamicvl@x{-2}}
7754 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7755   \edef\bbl@tempa{%
7756     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7757   \edef#5{%
7758     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7759   \edef#6{\fp_eval:n{
7760     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7761   \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).

```
7762 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7763   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7764   57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7765   57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7766   57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7767   58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7768   58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7769   58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7770   58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7771   59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7772   59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7773   59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7774   60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7775   60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7776   60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7777   60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7778   61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7779   61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7780   61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7781   62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7782   62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7783   62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7784   63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7785   63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7786   63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7787   63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7788   64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7789   64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7790   64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7791   65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7792   65401,65431,65460,65490,65520}
```

```

7793 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7794 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7795 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7796 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7797   \ifnum#2>2014 \ifnum#2<2038
7798     \bbl@afterfi\expandafter@gobble
7799   \fi\fi
7800   {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}}%
7801 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7802   \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7803 \count@\@ne
7804 \bbl@foreach\bbl@cs@umalqura@data{%
7805   \advance\count@\@ne
7806   \ifnum##1>\bbl@tempd\else
7807     \edef\bbl@tempe{\the\count@}%
7808     \edef\bbl@tempb{##1}%
7809     \fi}%
7810 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7811 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7812 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7813 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7814 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7815 \ExplSyntaxOff
7816 \bbl@add\bbl@precalendar{%
7817   \bbl@replace\bbl@ld@calendar{-civil}}}%
7818 \bbl@replace\bbl@ld@calendar{-umalqura}}}%
7819 \bbl@replace\bbl@ld@calendar{+}}}%
7820 \bbl@replace\bbl@ld@calendar{-}}}%
7821 \ca-islamic)

```

12.2 Hebrew

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

```

7822 (*ca-hebrew)
7823 \newcount\bbl@cntcommon
7824 \def\bbl@remainder#1#2#3{%
7825   #3=#1\relax
7826   \divide #3 by #2\relax
7827   \multiply #3 by -#2\relax
7828   \advance #3 by #1\relax}%
7829 \newif\ifbbl@divisible
7830 \def\bbl@checkifdivisible#1#2{%
7831   {\countdef\tmp=0
7832     \bbl@remainder{#1}{#2}{\tmp}%
7833     \ifnum \tmp=0
7834       \global\bbl@divisibletrue
7835     \else
7836       \global\bbl@divisiblefalse
7837     \fi}}
7838 \newif\ifbbl@gregleap
7839 \def\bbl@ifgregleap#1{%
7840   \bbl@checkifdivisible{#1}{4}%
7841   \ifbbl@divisible
7842     \bbl@checkifdivisible{#1}{100}%
7843     \ifbbl@divisible
7844       \bbl@checkifdivisible{#1}{400}%
7845       \ifbbl@divisible
7846         \bbl@gregleaptrue
7847       \else
7848         \bbl@gregleapfalse
7849     \fi

```

```

7850     \else
7851         \bbl@gregleaptrue
7852     \fi
7853 \else
7854     \bbl@gregleapfalse
7855 \fi
7856 \ifbbl@gregleap}
7857 \def\bbl@gregdayspriormonths#1#2#3{%
7858     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7859         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7860     \bbl@ifgregleap{#2}%
7861     \ifnum #1 > 2
7862         \advance #3 by 1
7863     \fi
7864     \fi
7865     \global\bbl@cntcommon=#3}%
7866     #3=\bbl@cntcommon}
7867 \def\bbl@gregdaysprioryears#1#2{%
7868     {\countdef\tmpc=4
7869     \countdef\tmpb=2
7870     \tmpb=#1\relax
7871     \advance \tmpb by -1
7872     \tmpc=\tmpb
7873     \multiply \tmpc by 365
7874     #2=\tmpc
7875     \tmpc=\tmpb
7876     \divide \tmpc by 4
7877     \advance #2 by \tmpc
7878     \tmpc=\tmpb
7879     \divide \tmpc by 100
7880     \advance #2 by -\tmpc
7881     \tmpc=\tmpb
7882     \divide \tmpc by 400
7883     \advance #2 by \tmpc
7884     \global\bbl@cntcommon=#2\relax}%
7885     #2=\bbl@cntcommon}
7886 \def\bbl@absfromgreg#1#2#3#4{%
7887     {\countdef\tmpd=0
7888     #4=#1\relax
7889     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7890     \advance #4 by \tmpd
7891     \bbl@gregdaysprioryears{#3}{\tmpd}%
7892     \advance #4 by \tmpd
7893     \global\bbl@cntcommon=#4\relax}%
7894     #4=\bbl@cntcommon}
7895 \newif\ifbbl@hebrleap
7896 \def\bbl@checkleaphebryear#1{%
7897     {\countdef\tmpa=0
7898     \countdef\tmpb=1
7899     \tmpa=#1\relax
7900     \multiply \tmpa by 7
7901     \advance \tmpa by 1
7902     \bbl@remainder{\tmpa}{19}{\tmpb}%
7903     \ifnum \tmpb < 7
7904         \global\bbl@hebrleaptrue
7905     \else
7906         \global\bbl@hebrleapfalse
7907     \fi}}
7908 \def\bbl@hebrlapsedmonths#1#2{%
7909     {\countdef\tmpa=0
7910     \countdef\tmpb=1
7911     \countdef\tmpc=2
7912     \tmpa=#1\relax

```

```

7913 \advance \tmpa by -1
7914 #2=\tmpa
7915 \divide #2 by 19
7916 \multiply #2 by 235
7917 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7918 \tmpc=\tmpb
7919 \multiply \tmpb by 12
7920 \advance #2 by \tmpb
7921 \multiply \tmpc by 7
7922 \advance \tmpc by 1
7923 \divide \tmpc by 19
7924 \advance #2 by \tmpc
7925 \global\bbl@cntcommon=#2}%
7926 #2=\bbl@cntcommon}
7927 \def\bbl@hebreleapseddays#1#2{%
7928 {\countdef\tmpa=0
7929 \countdef\tmpb=1
7930 \countdef\tmpc=2
7931 \bbl@hebreleapsedmonths{#1}{#2}%
7932 \tmpa=#2\relax
7933 \multiply \tmpa by 13753
7934 \advance \tmpa by 5604
7935 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7936 \divide \tmpa by 25920
7937 \multiply #2 by 29
7938 \advance #2 by 1
7939 \advance #2 by \tmpa
7940 \bbl@remainder{#2}{7}{\tmpa}%
7941 \ifnum \tmpc < 19440
7942 \ifnum \tmpc < 9924
7943 \else
7944 \ifnum \tmpa=2
7945 \bbl@checkleaphebyear{#1}% of a common year
7946 \ifbbl@hebrleap
7947 \else
7948 \advance #2 by 1
7949 \fi
7950 \fi
7951 \fi
7952 \ifnum \tmpc < 16789
7953 \else
7954 \ifnum \tmpa=1
7955 \advance #1 by -1
7956 \bbl@checkleaphebyear{#1}% at the end of leap year
7957 \ifbbl@hebrleap
7958 \advance #2 by 1
7959 \fi
7960 \fi
7961 \fi
7962 \else
7963 \advance #2 by 1
7964 \fi
7965 \bbl@remainder{#2}{7}{\tmpa}%
7966 \ifnum \tmpa=0
7967 \advance #2 by 1
7968 \else
7969 \ifnum \tmpa=3
7970 \advance #2 by 1
7971 \else
7972 \ifnum \tmpa=5
7973 \advance #2 by 1
7974 \fi
7975 \fi

```

```

7976 \fi
7977 \global\bbbl@cntcommon=#2\relax}%
7978 #2=\bbbl@cntcommon}
7979 \def\bbbl@daysinhebryear#1#2{%
7980 {\countdef\tmpe=12
7981 \bbbl@hebreleapseddays{#1}{\tmpe}%
7982 \advance #1 by 1
7983 \bbbl@hebreleapseddays{#1}{#2}%
7984 \advance #2 by -\tmpe
7985 \global\bbbl@cntcommon=#2}%
7986 #2=\bbbl@cntcommon}
7987 \def\bbbl@hebrdayspriormonths#1#2#3{%
7988 {\countdef\tmpf= 14
7989 #3=\ifcase #1\relax
7990 0 \or
7991 0 \or
7992 30 \or
7993 59 \or
7994 89 \or
7995 118 \or
7996 148 \or
7997 148 \or
7998 177 \or
7999 207 \or
8000 236 \or
8001 266 \or
8002 295 \or
8003 325 \or
8004 400
8005 \fi
8006 \bbbl@checkleaphebryear{#2}%
8007 \ifbbbl@hebrleap
8008 \ifnum #1 > 6
8009 \advance #3 by 30
8010 \fi
8011 \fi
8012 \bbbl@daysinhebryear{#2}{\tmpf}%
8013 \ifnum #1 > 3
8014 \ifnum \tmpf=353
8015 \advance #3 by -1
8016 \fi
8017 \ifnum \tmpf=383
8018 \advance #3 by -1
8019 \fi
8020 \fi
8021 \ifnum #1 > 2
8022 \ifnum \tmpf=355
8023 \advance #3 by 1
8024 \fi
8025 \ifnum \tmpf=385
8026 \advance #3 by 1
8027 \fi
8028 \fi
8029 \global\bbbl@cntcommon=#3\relax}%
8030 #3=\bbbl@cntcommon}
8031 \def\bbbl@absfromhebr#1#2#3#4{%
8032 {#4=#1\relax
8033 \bbbl@hebrdayspriormonths{#2}{#3}{#1}%
8034 \advance #4 by #1\relax
8035 \bbbl@hebreleapseddays{#3}{#1}%
8036 \advance #4 by #1\relax
8037 \advance #4 by -1373429
8038 \global\bbbl@cntcommon=#4\relax}%

```

```

8039 #4=\bbl@cntcommon}
8040 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8041 {\countdef\tmpx= 17
8042 \countdef\tmpy= 18
8043 \countdef\tmpz= 19
8044 #6=#3\relax
8045 \global\advance #6 by 3761
8046 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8047 \tmpz=1 \tmpy=1
8048 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8049 \ifnum \tmpx > #4\relax
8050 \global\advance #6 by -1
8051 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8052 \fi
8053 \advance #4 by -\tmpx
8054 \advance #4 by 1
8055 #5=#4\relax
8056 \divide #5 by 30
8057 \loop
8058 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8059 \ifnum \tmpx < #4\relax
8060 \advance #5 by 1
8061 \tmpy=\tmpx
8062 \repeat
8063 \global\advance #5 by -1
8064 \global\advance #4 by -\tmpy}}
8065 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8066 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8067 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8068 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8069 \bbl@hebrfromgreg
8070 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8071 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8072 \edef#4{\the\bbl@hebyear}%
8073 \edef#5{\the\bbl@hebrmonth}%
8074 \edef#6{\the\bbl@hebrday}}
8075 \ca-hebrew

```

12.3 Persian

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

```

8076 \ca-persian
8077 \ExplSyntaxOn
8078 \langle\Compute Julian day\rangle
8079 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8080 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8081 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8082 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8083 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8084 \bbl@afterfi\expandafter@gobble
8085 \fi\fi
8086 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}}%
8087 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8088 \ifin@{\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8089 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8090 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8091 \ifnum\bbl@tempc<\bbl@tempb
8092 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8093 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%

```



```

8094 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8095 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8096 \fi
8097 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8098 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8099 \edef#5{\fp_eval:n{% set Jalali month
8100   (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8101 \edef#6{\fp_eval:n{% set Jalali day
8102   (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}%
8103 \ExplSyntaxOff
8104 \ca-persian)

```

12.4 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.

```

8105 (*ca-coptic)
8106 \ExplSyntaxOn
8107 <<Compute Julian day>>
8108 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8109   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8110   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8111   \edef#4{\fp_eval:n{%
8112     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8113   \edef\bbl@tempc{\fp_eval:n{%
8114     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8115   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8116   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8117 \ExplSyntaxOff
8118 \ca-coptic)
8119 (*ca-ethiopic)
8120 \ExplSyntaxOn
8121 <<Compute Julian day>>
8122 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8123   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8124   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8125   \edef#4{\fp_eval:n{%
8126     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8127   \edef\bbl@tempc{\fp_eval:n{%
8128     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8129   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8130   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}%
8131 \ExplSyntaxOff
8132 \ca-ethiopic)

```

12.5 Buddhist

That's very simple.

```

8133 (*ca-buddhist)
8134 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8135   \edef#4{\number\numexpr#1+543\relax}%
8136   \edef#5{#2}%
8137   \edef#6{#3}%
8138 \ca-buddhist)

```

13 Support for Plain T_EX (`plain.def`)

13.1 Not renaming `hyphen.tex`

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to

load hyphenation patterns for other languages in a plain-based T_EX-format. When asked he responded:

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

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

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

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

```

8139 \catcode\begin=1 % left brace is begin-group character
8140 \catcode\end=2 % right brace is end-group character
8141 \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).

```
8143 \openin 0 hyphen.cfg
8144 \ifeof0
8145 \else
8146   \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.

```

8147 \def\input #1 {%
8148     \let\input\@
8149     \a hyphen.cfg
8150     \let\@undefined
8151 }
8152 \fi
8153 </bplain | bplain>

```

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

```
8154 <bplain>\a plain.tex
8155 <blplain>\a lplain.tex
```

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

```
8156 \def\fmtname{babel-plain}
8157 \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.

13.2 Emulating some L^AT_EX 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 no 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).

```

8158 <<*Emulate LaTeX>> ≡
8159 \def\@empty{}
8160 \def\loadlocalcfg#1{%
8161   \openin0#1.cfg
8162   \ifeof0
8163     \closein0
8164   \else

```

```

8165 \closein0
8166 {\immediate\writel6{*****}%
8167 \immediate\writel6{* Local config file #1.cfg used}%
8168 \immediate\writel6{*}%
8169 }
8170 \input #1.cfg\relax
8171 \fi
8172 \@endofldf}

```

13.3 General tools

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

```

8173 \long\def\@firstofone#1{#1}
8174 \long\def\@firstoftwo#1#2{#1}
8175 \long\def\@secondoftwo#1#2{#2}
8176 \def\@nnil{\@nil}
8177 \def\@gobbletwo#1#2{}
8178 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8179 \def\@star@or@long#1{%
8180 \ifstar
8181 {\let\l@ngrel@x\relax#1}%
8182 {\let\l@ngrel@x\long#1}}
8183 \let\l@ngrel@x\relax
8184 \def\@car#1#2\@nil{#1}
8185 \def\@cdr#1#2\@nil{#2}
8186 \let\@typeset@protect\relax
8187 \let\protected@edef\edef
8188 \long\def\@gobble#1{}
8189 \edef\@backslashchar{\expandafter\@gobble\string\}
8190 \def\strip@prefix#1>{}
8191 \def\g@addto@macro#1#2{{%
8192 \toks@{\expandafter{#1#2}%
8193 \xdef#1{\the\toks@}}}
8194 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8195 \def\@nameuse#1{\csname #1\endcsname}
8196 \def\@ifundefined#1{%
8197 \expandafter\ifx\csname#1\endcsname\relax
8198 \expandafter\@firstoftwo
8199 \else
8200 \expandafter\@secondoftwo
8201 \fi}
8202 \def\@expandtwoargs#1#2#3{%
8203 \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8204 \def\zap@space#1 #2{%
8205 #1%
8206 \ifx#2\@empty\else\expandafter\zap@space\fi
8207 #2}
8208 \let\bbl@trace\@gobble
8209 \def\bbl@error#1#2{%
8210 \begingroup
8211 \newlinechar=`^^J
8212 \def\{^^J(babel) }%
8213 \errhelp{#2}\errmessage{\#1}%
8214 \endgroup}
8215 \def\bbl@warning#1{%
8216 \begingroup
8217 \newlinechar=`^^J
8218 \def\{^^J(babel) }%
8219 \message{\#1}%
8220 \endgroup}
8221 \let\bbl@infowarn\bbl@warning
8222 \def\bbl@info#1{%
8223 \begingroup

```

```

8224 \newlinechar=`^^J
8225 \def\{\{^J}%
8226 \wlog{#1}%
8227 \endgroup}

```

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

```

8228 \ifx\@preamblecmds\undefined
8229 \def\@preamblecmds{}
8230 \fi
8231 \def\@onlypreamble#1{%
8232 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8233 \@preamblecmds\do#1}}
8234 \@onlypreamble\@onlypreamble

```

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

```

8235 \def\begindocument{%
8236 \@begindocumenthook
8237 \global\let\@begindocumenthook\undefined
8238 \def\do##1{\global\let##1\undefined}%
8239 \@preamblecmds
8240 \global\let\do\noexpand}

8241 \ifx\@begindocumenthook\undefined
8242 \def\@begindocumenthook{}
8243 \fi
8244 \@onlypreamble\@begindocumenthook
8245 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

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

```

8246 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8247 \@onlypreamble\AtEndOfPackage
8248 \def\@endofldf{}
8249 \@onlypreamble\@endofldf
8250 \let\bbl@afterlang\empty
8251 \chardef\bbl@opt@hyphenmap\z@

```

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

```

8252 \catcode`\&=\z@
8253 \ifx&if@filesw\undefined
8254 \expandafter\let\csname if@filesw\expandafter\endcsname
8255 \csname iffalse\endcsname
8256 \fi
8257 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

8258 \def\newcommand{\@star@or@long\new@command}
8259 \def\new@command#1{%
8260 \@testopt{\@newcommand#1}0}
8261 \def\@newcommand#1[#2]{%
8262 \@ifnextchar [{\@xargdef#1[#2]}%
8263 {\@argdef#1[#2]}}
8264 \long\def\@argdef#1[#2]#3{%
8265 \@yargdef#1\@ne{#2}{#3}}
8266 \long\def\@xargdef#1[#2]#3#4{%
8267 \expandafter\def\expandafter#1\expandafter{%
8268 \expandafter\@protected@testopt\expandafter #1%
8269 \csname\string#1\expandafter\endcsname{#3}}%
8270 \expandafter\@yargdef \csname\string#1\endcsname
8271 \tw@{#2}{#4}}
8272 \long\def\@yargdef#1#2#3{%

```

```

8273 \@tempcnta#3\relax
8274 \advance \@tempcnta \@ne
8275 \let\@hash@\relax
8276 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8277 \@tempcntb #2%
8278 \@whilenum\@tempcntb <\@tempcnta
8279 \do{%
8280   \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8281   \advance\@tempcntb \@ne}%
8282 \let\@hash@##%
8283 \l@ngrel\@x\expandafter\def\expandafter#1\reserved@a}
8284 \def\providecommand{\@star@or@long\provide@command}
8285 \def\provide@command#1{%
8286   \begingroup
8287     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8288   \endgroup
8289   \expandafter\@ifundefined\@gtempa
8290     {\def\reserved@a{\new@command#1}}%
8291     {\let\reserved@a\relax
8292      \def\reserved@a{\new@command\reserved@a}}%
8293   \reserved@a}%
8294 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8295 \def\declare@robustcommand#1{%
8296   \edef\reserved@a{\string#1}%
8297   \def\reserved@b{#1}%
8298   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8299   \edef#1{%
8300     \ifx\reserved@a\reserved@b
8301       \noexpand\x@protect
8302       \noexpand#1%
8303     \fi
8304     \noexpand\protect
8305     \expandafter\noexpand\csname
8306       \expandafter\@gobble\string#1 \endcsname
8307   }%
8308   \expandafter\new@command\csname
8309     \expandafter\@gobble\string#1 \endcsname
8310 }
8311 \def\x@protect#1{%
8312   \ifx\protect\@typeset@protect\else
8313     \@x@protect#1%
8314   \fi
8315 }
8316 \catcode`\&=\z@ % Trick to hide conditionals
8317 \def\@x@protect#1&#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`.

```

8318 \def\bbl@tempa{\csname newif\endcsname&fin@}
8319 \catcode`\&=4
8320 \ifx\in@\@undefined
8321   \def\in@#1#2{%
8322     \def\in@@##1##2##3\in@{%
8323       \ifx\in@##2\in@false\else\in@true\fi}%
8324     \in@@#2#1\in@\in@@}
8325 \else
8326   \let\bbl@tempa\@empty
8327 \fi
8328 \bbl@tempa

```

\TeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and

activeacute). For plain \TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```
8329 \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.

```
8330 \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.

```
8331 \ifx\@tempcnta\undefined
8332   \csname newcount\endcsname\@tempcnta\relax
8333 \fi
8334 \ifx\@tempcntb\undefined
8335   \csname newcount\endcsname\@tempcntb\relax
8336 \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`).

```
8337 \ifx\bye\undefined
8338   \advance\count10 by -2\relax
8339 \fi
8340 \ifx\@ifnextchar\undefined
8341   \def\@ifnextchar#1#2#3{%
8342     \let\reserved@d=#1%
8343     \def\reserved@a{#2}\def\reserved@b{#3}%
8344     \futurelet\@let@token\@ifnch}
8345   \def\@ifnch{%
8346     \ifx\@let@token\@sptoken
8347       \let\reserved@c\@xifnch
8348     \else
8349       \ifx\@let@token\reserved@d
8350         \let\reserved@c\reserved@a
8351       \else
8352         \let\reserved@c\reserved@b
8353     \fi
8354   \fi
8355   \reserved@c}
8356   \def\{\let\@sptoken= }\: % this makes \@sptoken a space token
8357   \def\{\@xifnch} \expandafter\def\{\futurelet\@let@token\@ifnch}
8358 \fi
8359 \def\@testopt#1#2{%
8360   \@ifnextchar[#{#1}{#1[#2]}}
8361 \def\@protected@testopt#1{%
8362   \ifx\protect\@typeset@protect
8363     \expandafter\@testopt
8364   \else
8365     \@x@protect#1%
8366   \fi}
8367 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8368   #2\relax}\fi}
8369 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8370   \else\expandafter\@gobble\fi{#1}}
```

13.4 Encoding related macros

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

```
8371 \def\DeclareTextCommand{%
8372   \@dec@text@cmd\providecommand
8373 }
8374 \def\ProvideTextCommand{%
8375   \@dec@text@cmd\providecommand
```

```

8376 }
8377 \def\DeclareTextSymbol#1#2#3{%
8378   \@dec@text@cmd\chardef#1{#2}#3\relax
8379 }
8380 \def\@dec@text@cmd#1#2#3{%
8381   \expandafter\def\expandafter#2%
8382     \expandafter{%
8383       \csname#3-cmd\expandafter\endcsname
8384       \expandafter#2%
8385       \csname#3\string#2\endcsname
8386     }%
8387 %   \let\@ifdefinable\@rc@ifdefinable
8388   \expandafter#1\csname#3\string#2\endcsname
8389 }
8390 \def\@current@cmd#1{%
8391   \ifx\protect\@typeset@protect\else
8392     \noexpand#1\expandafter\@gobble
8393   \fi
8394 }
8395 \def\@changed@cmd#1#2{%
8396   \ifx\protect\@typeset@protect
8397     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8398       \expandafter\ifx\csname ?\string#1\endcsname\relax
8399         \expandafter\def\csname ?\string#1\endcsname{%
8400           \@changed@x@err{#1}%
8401         }%
8402       \fi
8403     \global\expandafter\let
8404       \csname\cf@encoding\string#1\expandafter\endcsname
8405       \csname ?\string#1\endcsname
8406   \fi
8407   \csname\cf@encoding\string#1%
8408     \expandafter\endcsname
8409   \else
8410     \noexpand#1%
8411   \fi
8412 }
8413 \def\@changed@x@err#1{%
8414   \errhelp{Your command will be ignored, type <return> to proceed}%
8415   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8416 \def\DeclareTextCommandDefault#1{%
8417   \DeclareTextCommand#1?%
8418 }
8419 \def\ProvideTextCommandDefault#1{%
8420   \ProvideTextCommand#1?%
8421 }
8422 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8423 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8424 \def\DeclareTextAccent#1#2#3{%
8425   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8426 }
8427 \def\DeclareTextCompositeCommand#1#2#3#4{%
8428   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8429   \edef\reserved@b{\string##1}%
8430   \edef\reserved@c{%
8431     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8432   \ifx\reserved@b\reserved@c
8433     \expandafter\expandafter\expandafter\ifx
8434       \expandafter\@car\reserved@a\relax\relax\@nil
8435     \@text@composite
8436   \else
8437     \edef\reserved@b##1{%
8438       \def\expandafter\noexpand

```

```

8439         \csname#2\string#1\endcsname###1{%
8440         \noexpand\@text@composite
8441         \expandafter\noexpand\csname#2\string#1\endcsname
8442         ###1\noexpand\@empty\noexpand\@text@composite
8443         {##1}%
8444     }%
8445 }%
8446 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8447 \fi
8448 \expandafter\def\csname\expandafter\string\csname
8449     #2\endcsname\string#1-\string#3\endcsname{#4}
8450 \else
8451     \errhelp{Your command will be ignored, type <return> to proceed}%
8452     \errmessage{\string\DeclareTextCompositeCommand\space used on
8453         inappropriate command \protect#1}
8454 \fi
8455 }
8456 \def\@text@composite#1#2#3\@text@composite{%
8457     \expandafter\@text@composite@x
8458     \csname\string#1-\string#2\endcsname
8459 }
8460 \def\@text@composite@x#1#2{%
8461     \ifx#1\relax
8462         #2%
8463     \else
8464         #1%
8465     \fi
8466 }
8467 %
8468 \def\@strip@args#1:#2-#3\@strip@args{#2}
8469 \def\DeclareTextComposite#1#2#3#4{%
8470     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8471     \bgroup
8472     \lccode`\@=#4%
8473     \lowercase{%
8474     \egroup
8475     \reserved@a @%
8476     }%
8477 }
8478 %
8479 \def\UseTextSymbol#1#2{#2}
8480 \def\UseTextAccent#1#2#3{}
8481 \def\@use@text@encoding#1{}
8482 \def\DeclareTextSymbolDefault#1#2{%
8483     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8484 }
8485 \def\DeclareTextAccentDefault#1#2{%
8486     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8487 }
8488 \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.

```

8489 \DeclareTextAccent{"}{OT1}{127}
8490 \DeclareTextAccent{'}{OT1}{19}
8491 \DeclareTextAccent{^}{OT1}{94}
8492 \DeclareTextAccent{`}{OT1}{18}
8493 \DeclareTextAccent{~}{OT1}{126}

```

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

```

8494 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8495 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8496 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8497 \DeclareTextSymbol{\textquoteright}{OT1}{``'}

```



```

8498 \DeclareTextSymbol{\i}{OT1}{16}
8499 \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`.

```

8500 \ifx\scriptsize\@undefined
8501   \let\scriptsize\sevenrm
8502 \fi

```

And a few more “dummy” definitions.

```

8503 \def\language{english}%
8504 \let\bbl@opt@shorthands\@nnil
8505 \def\bbl@ifshorthand#1#2#3{#2}%
8506 \let\bbl@language@opts\@empty
8507 \let\bbl@ensureinfo\@gobble
8508 \let\bbl@provide@locale\relax
8509 \ifx\babeloptionstrings\@undefined
8510   \let\bbl@opt@strings\@nnil
8511 \else
8512   \let\bbl@opt@strings\babeloptionstrings
8513 \fi
8514 \def\BabelStringsDefault{generic}
8515 \def\bbl@tempa{normal}
8516 \ifx\babeloptionmath\bbl@tempa
8517   \def\bbl@mathnormal{\noexpand\textormath}
8518 \fi
8519 \def\AfterBabelLanguage#1#2{}
8520 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8521 \let\bbl@afterlang\relax
8522 \def\bbl@opt@safe{BR}
8523 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
8524 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8525 \expandafter\newif\csname ifbbl@single\endcsname
8526 \chardef\bbl@bidimode\z@
8527 <\/Emulate LaTeX>

```

A proxy file:

```

8528 <*\plain>
8529 \input babel.def
8530 </\plain>

```

14 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 \LaTeX 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, *\LaTeX , A document preparation System*, Addison-Wesley, 1986.

- [7] Leslie Lamport, in: *T_EXhax Digest*, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018
- [10] Hubert Partl, *German T_EX*, *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International L^AT_EX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using L^AT_EX*, Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer; een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).