

Babel

Code

Version 3.93.24338
2023/08/31

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

T_EX

pdfT_EX

LuaT_EX

XeT_EX

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	50
4.12.2	Letters	51
4.12.3	Shorthands for quotation marks	52
4.12.4	Umlauts and tremas	53
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	83
5.3.1	<code>ifthen</code>	83
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	101
9.1	XeTeX	101
9.2	Layout	103
9.3	8-bit TeX	104
9.4	LuaTeX	105
9.5	Southeast Asian scripts	111
9.6	CJK line breaking	113

9.7	Arabic justification	115
9.8	Common stuff	119
9.9	Automatic fonts and ids switching	119
9.10	Bidi	125
9.11	Layout	127
9.12	Lua: transforms	134
9.13	Lua: Auto bidi with basic and basic-r	143
10	Data for CJK	154
11	The ‘nil’ language	154
12	Calendars	155
12.1	Islamic	155
12.2	Hebrew	157
12.3	Persian	161
12.4	Coptic and Ethiopic	161
12.5	Buddhist	162
13	Support for Plain T_EX (plain.def)	163
13.1	Not renaming hyphen.tex	163
13.2	Emulating some L ^A T _E X features	164
13.3	General tools	164
13.4	Encoding related macros	168
14	Acknowledgements	171

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=3.93.24338>>
2 <<date=2023/08/31>>
```

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

```

3.1 Multiple languages

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

```

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

```



```

200 \ifx\language\@undefined
201   \csname newcount\endcsname\language
202 \fi
203 <</Define core switching macros>>

```

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

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

```

204 <<*Define core switching macros>> ≡
205 \countdef\last@language=19
206 \def\addlanguage{\csname newlanguage\endcsname}
207 <</Define core switching macros>>

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it). Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

3.2 The Package File (\LaTeX , `babel.sty`)

```

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

```

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

```

211 \ifpackagewith{babel}{debug}
212   {\providecommand\bbbl@trace[1]{\message{^^J[ #1 ]}}%
213    \let\bbbl@debug\@firstofone
214    \ifx\directlua\@undefined\else
215      \directlua{ Babel = Babel or {}
216        Babel.debug = true }%
217      \input{babel-debug.tex}%
218    \fi}
219 {\providecommand\bbbl@trace[1]{}%
220  \let\bbbl@debug@gobble
221  \ifx\directlua\@undefined\else
222    \directlua{ Babel = Babel or {}
223      Babel.debug = false }%
224  \fi}
225 \def\bbbl@error#1#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#1.1df-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}{\@noletter{#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\@#5{%
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\@#5\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\@#5\bbl@tempb
508 \bbl@bcpcase#3\@empty\@empty\@#5\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@initload\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@bcpoptions]{\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\@bbl@hyphenmap\endcsname
729 \fi
730 \chardef\bbl@opt@hyphenmap\z@
731 \else
732 \ifnum\bbl@hymap>\bbl@opt@hyphenmap\else
733 \csname\language\@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 `\(lang)hyphenmins` 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 $\text{\LaTeX 2}_{\epsilon}$. When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by babel. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

```

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 $\text{\LaTeX 2}_{\epsilon}$, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

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

```

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 locale name
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   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1171 \<core>
1172   \ifx\bbl@normalsf\@empty
1173     \ifnum\sfcodes\@frenchspacing
1174       \let\normalsfcodes\@frenchspacing
1175     \else
1176       \let\normalsfcodes\@nonfrenchspacing
1177     \fi
1178   \else
1179     \let\normalsfcodes\bbl@normalsf
1180   \fi
1181 \<+core>
1182   \ifbbl@single % must go after the line above.
1183     \renewcommand\selectlanguage[1]{}%
1184     \renewcommand\foreignlanguage[2]{#2}%
1185     \global\let\babel@aux\@gobbletwo % Also as flag
1186   \fi}
1187 \<core>
1188 \AddToHook{begindocument/before}{%
1189   \let\bbl@normalsf\normalsfcodes
1190   \let\normalsfcodes\relax} % Hack, to delay the setting
1191 \<+core>
1192 \ifcase\bbl@engine\or
1193   \AtBeginDocument{\pagedir\bodydir} % TODO - a better place
1194 \fi

```

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

```

1195 \def\select@language@x#1{%
1196   \ifcase\bbl@select@type
1197     \bbl@ifsamestring\language{#1}{\select@language{#1}}%
1198   \else
1199     \select@language{#1}%
1200   \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.

```

1201 \bbl@trace{Shorhands}
1202 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1203   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1204   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1205   \ifx\nfss@catcodes\undefined\else % TODO - same for above
1206     \begingroup
1207       \catcode`#1\active
1208       \nfss@catcodes
1209       \ifnum\catcode`#1=\active
1210         \endgroup
1211         \bbl@add\nfss@catcodes{\@makeother#1}%
1212       \else
1213         \endgroup
1214       \fi
1215   \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.

```

1216 \def\bbl@remove@special#1{%
1217   \begingroup
1218     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1219       \else\noexpand##1\noexpand##2\fi}%
1220     \def\do{\x\do}%
1221     \def\@makeother{\x\@makeother}%
1222   \edef\x{\endgroup
1223     \def\noexpand\dospecials{\dospecials}%
1224     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1225       \def\noexpand\@sanitize{\@sanitize}%
1226     \fi}%
1227   \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).

```

1228 \def\bbl@active@def#1#2#3#4{%
1229   \@namedef{#3#1}{%
1230     \expandafter\ifx\csname#2@sh#1\endcsname\relax
1231       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1232     \else
1233       \bbl@afterfi\csname#2@sh#1\endcsname
1234     \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.

```

1235 \long\@namedef{#3@arg#1}##1{%
1236   \expandafter\ifx\csname#2@sh#1@\string##1@\endcsname\relax
1237     \bbl@afterelse\csname#4#1\endcsname##1%
1238   \else
1239     \bbl@afterfi\csname#2@sh#1@\string##1@\endcsname
1240   \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.

```

1241 \def\initiate@active@char#1{%
1242   \bbl@ifunset{active@char\string#1}%
1243   {\bbl@withactive
1244     {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1245   {}}

```

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).

```

1246 \def\@initiate@active@char#1#2#3{%
1247   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1248   \ifx#1\@undefined
1249     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1250   \else
1251     \bbl@csarg\let{oridef@@#2}#1%
1252     \bbl@csarg\edef{oridef@#2}{%
1253       \let\noexpand#1%
1254       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1255   \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*").

```

1256 \ifx#1#3\relax
1257   \expandafter\let\csname normal@char#2\endcsname#3%
1258 \else
1259   \bbl@info{Making #2 an active character}%
1260   \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1261     \@namedef{normal@char#2}{%
1262       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1263   \else
1264     \@namedef{normal@char#2}{#3}%
1265   \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).

```

1266 \bbl@restoreactive{#2}%
1267 \AtBeginDocument{%
1268   \catcode`#2\active
1269   \if@filesw
1270     \immediate\write\@mainaux{\catcode`\string#2\active}%
1271   \fi}%
1272 \expandafter\bbl@add@special\csname#2\endcsname
1273 \catcode`#2\active
1274 \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⟩`).

```

1275 \let\bbl@tempa\@firstoftwo
1276 \if\string^#2%
1277   \def\bbl@tempa{\noexpand\textormath}%
1278 \else
1279   \ifx\bbl@mathnormal\@undefined\else
1280     \let\bbl@tempa\bbl@mathnormal
1281   \fi
1282 \fi
1283 \expandafter\edef\csname active@char#2\endcsname{%
1284   \bbl@tempa
1285     {\noexpand\if@safe@actives
1286       \noexpand\expandafter
1287       \expandafter\noexpand\csname normal@char#2\endcsname
1288     \noexpand\else
1289       \noexpand\expandafter
1290       \expandafter\noexpand\csname bbl@doactive#2\endcsname
1291     \noexpand\fi}%
1292   {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1293 \bbl@csarg\edef{doactive#2}{%
1294   \expandafter\noexpand\csname user@active#2\endcsname}%

```

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

`\active@prefix⟨char⟩\normal@char⟨char⟩`

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

```

1295 \bbl@csarg\edef{active@#2}{%
1296   \noexpand\active@prefix\noexpand#1%
1297   \expandafter\noexpand\csname active@char#2\endcsname}%
1298 \bbl@csarg\edef{normal@#2}{%
1299   \noexpand\active@prefix\noexpand#1%
1300   \expandafter\noexpand\csname normal@char#2\endcsname}%
1301 \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.

```

1302 \bbl@active@def#2\user@group{user@active}{language@active}%
1303 \bbl@active@def#2\language@group{language@active}{system@active}%
1304 \bbl@active@def#2\system@group{system@active}{normal@char}%

```

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

```

1305 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1306   {\expandafter\noexpand\csname normal@char#2\endcsname}%
1307 \expandafter\edef\csname\user@group @sh#2@\string\protect@\endcsname
1308   {\expandafter\noexpand\csname user@active#2\endcsname}%

```

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (‘) active we need to change `\pr@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.

```

1309 \if\string'#2%
1310   \let\prim@s\bbl@prim@s
1311   \let\active@math@prime#1%
1312 \fi
1313 \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}

```

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

```

1314 <<{*More package options}>> ≡
1315 \DeclareOption{math=active}{}
1316 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1317 <</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*.

```

1318 \@ifpackagewith{babel}{KeepShorthandsActive}%
1319   {\let\bbl@restoreactive@gobble}%
1320   {\def\bbl@restoreactive#1{%
1321     \bbl@exp{%
1322       \\\AfterBabelLanguage\\CurrentOption
1323       {\catcode`#1=\the\catcode`#1\relax}%
1324       \\\AtEndOfPackage
1325       {\catcode`#1=\the\catcode`#1\relax}}}%
1326   \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.

```

1327 \def\bbl@sh@select#1#2{%
1328   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1329     \bbl@afterelse\bbl@scndcs
1330   \else
1331     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1332   \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 `\protects` the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar`: (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsname` is available. If there is, the expansion will be more robust.

```

1333 \begingroup
1334 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1335   {\gdef\active@prefix#1{%
1336     \ifx\protect\@typeset@protect
1337     \else
1338       \ifx\protect\@unexpandable@protect
1339         \noexpand#1%
1340       \else
1341         \protect#1%
1342       \fi
1343     \expandafter\@gobble
1344     \fi}}
1345   {\gdef\active@prefix#1{%
1346     \ifincsname
1347       \string#1%
1348       \expandafter\@gobble
1349     \else
1350       \ifx\protect\@typeset@protect
1351       \else
1352         \ifx\protect\@unexpandable@protect
1353           \noexpand#1%
1354         \else
1355           \protect#1%
1356         \fi
1357       \expandafter\expandafter\expandafter\@gobble
1358       \fi

```

```

1359     \fi}}
1360 \endgroup

```

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

```

1361 \newif\if@safe@actives
1362 \@safe@activefalse

```

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

```

1363 \def\bbl@restore@actives{\if@safe@actives\@safe@activefalse\fi}

```

\bbl@activate Both macros take one argument, like `\initiate@active@char`. The macro is used to change the definition of an active character to expand to `\active@char⟨char⟩` in the case of `\bbl@activate`, or `\normal@char⟨char⟩` in the case of `\bbl@deactivate`.

```

1364 \chardef\bbl@activated\z@
1365 \def\bbl@activate#1{%
1366   \chardef\bbl@activated\@ne
1367   \bbl@withactive{\expandafter\let\expandafter}#1%
1368   \csname bbl@active@\string#1\endcsname}
1369 \def\bbl@deactivate#1{%
1370   \chardef\bbl@activated\tw@
1371   \bbl@withactive{\expandafter\let\expandafter}#1%
1372   \csname bbl@normal@\string#1\endcsname}

```

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

```

\bbl@scndcs
1373 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1374 \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.

```

1375 \def\babel@texpdf#1#2#3#4{%
1376   \ifx\texorpdfstring\@undefined
1377     \textormath{#1}{#3}%
1378   \else
1379     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1380     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1381   \fi}
1382 %
1383 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1384 \def\@decl@short#1#2#3\@nil#4{%
1385   \def\bbl@tempa{#3}%
1386   \ifx\bbl@tempa\@empty
1387     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1388     \bbl@ifunset{#1@sh@\string#2@}{}%
1389     {\def\bbl@tempa{#4}%
1390      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa

```

```

1391     \else
1392     \bbl@info
1393     {Redefining #1 shorthand \string#2\\%
1394     in language \CurrentOption}%
1395     \fi}%
1396     \@namedef{#1@sh@\string#2@}{#4}%
1397 \else
1398     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1399     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1400     {\def\bbl@tempa{#4}%
1401     \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1402     \else
1403     \bbl@info
1404     {Redefining #1 shorthand \string#2\string#3\\%
1405     in language \CurrentOption}%
1406     \fi}%
1407     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1408     \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.

```

1409 \def\textormath{%
1410   \ifmmode
1411     \expandafter\@secondoftwo
1412   \else
1413     \expandafter\@firstoftwo
1414   \fi}

```

`\user@group` The current concept of ‘shorthands’ supports three levels or groups of shorthands. For each level the name of the level or group is stored in a macro. The default is to have a user group; use language `\language@group` group ‘english’ and have a system group called ‘system’.

```

1415 \def\user@group{user}
1416 \def\language@group{english} % TODO. I don't like defaults
1417 \def\system@group{system}

```

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

```

1418 \def\useshorthands{%
1419   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}
1420 \def\bbl@usesh@s#1{%
1421   \bbl@usesh@x
1422   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1423   {#1}}
1424 \def\bbl@usesh@x#1#2{%
1425   \bbl@ifshorthand{#2}%
1426   {\def\user@group{user}%
1427    \initiate@active@char{#2}%
1428    #1%
1429    \bbl@activate{#2}}%
1430   {\bbl@error
1431    {I can't declare a shorthand turned off (\string#2)}
1432    {Sorry, but you can't use shorthands which have been\\%
1433     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 `\bbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1434 \def\user@language@group{user@\language@group}
1435 \def\bbl@set@user@generic#1#2{%

```

```

1436 \bbl@ifunset{user@generic@active#1}%
1437 {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1438 \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1439 \expandafter\edef\csname#2@sh@#1@\endcsname{%
1440 \expandafter\noexpand\csname normal@char#1\endcsname}%
1441 \expandafter\edef\csname#2@sh@#1@\string\protect\endcsname{%
1442 \expandafter\noexpand\csname user@active#1\endcsname}}%
1443 \@empty}
1444 \newcommand\defineshorthand[3][user]{%
1445 \edef\bbl@tempa{\zap@space#1 \@empty}%
1446 \bbl@for\bbl@tempb\bbl@tempa{%
1447 \if*\expandafter\@car\bbl@tempb\@nil
1448 \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1449 \@expandtwoargs
1450 \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1451 \fi
1452 \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].

```

1453 \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`.

```

1454 \def\aliasshorthand#1#2{%
1455 \bbl@ifshorthand{#2}%
1456 {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1457 \ifx\document\@notprerr
1458 \@notshorthand{#2}%
1459 \else
1460 \initiate@active@char{#2}%
1461 \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1462 \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1463 \bbl@activate{#2}%
1464 \fi
1465 \fi}%
1466 {\bbl@error
1467 {Cannot declare a shorthand turned off (\string#2)}
1468 {Sorry, but you cannot use shorthands which have been\\%
1469 turned off in the package options}}}

```

`\@notshorthand`

```

1470 \def\@notshorthand#1{%
1471 \bbl@error{%
1472 The character '\string #1' should be made a shorthand character;\\%
1473 add the command \string\usesshorthands\string{#1\string} to
1474 the preamble.\\%
1475 I will ignore your instruction}%
1476 {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.

```

1477 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1478 \DeclareRobustCommand*\shorthandoff{%
1479 \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1480 \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.

```

1481 \def\bbl@switch@sh#1#2{%
1482   \ifx#2\@nnil\else
1483     \bbl@ifunset{bbl@active@\string#2}%
1484     {\bbl@error
1485       {I can't switch '\string#2' on or off--not a shorthand}%
1486       {This character is not a shorthand. Maybe you made\\%
1487         a typing mistake? I will ignore your instruction.}}}%
1488     {\ifcase#1%   off, on, off*
1489       \catcode`#2\relax
1490       \or
1491       \catcode`#2\active
1492       \bbl@ifunset{bbl@shdef@\string#2}%
1493       {}%
1494       {\bbl@withactive{\expandafter\let\expandafter}#2%
1495         \csname bbl@shdef@\string#2\endcsname
1496         \bbl@csarg\let{shdef@\string#2}\relax}%
1497       \ifcase\bbl@activated\or
1498       \bbl@activate{#2}%
1499       \else
1500       \bbl@deactivate{#2}%
1501       \fi
1502       \or
1503       \bbl@ifunset{bbl@shdef@\string#2}%
1504       {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1505       {}%
1506       \csname bbl@oricat@\string#2\endcsname
1507       \csname bbl@oridef@\string#2\endcsname
1508       \fi}%
1509   \bbl@afterfi\bbl@switch@sh#1%
1510 \fi}

```

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

```

1511 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1512 \def\bbl@putsh#1{%
1513   \bbl@ifunset{bbl@active@\string#1}%
1514   {\bbl@putsh@i#1\@empty\@nnil}%
1515   {\csname bbl@active@\string#1\endcsname}}
1516 \def\bbl@putsh@i#1#2\@nnil{%
1517   \csname\language@group @sh@\string#1@%
1518   \ifx\@empty#2\else\string#2@\fi\endcsname}
1519 %
1520 \ifx\bbl@opt@shorthands\@nnil\else
1521   \let\bbl@s@initiate@active@char\initiate@active@char
1522   \def\initiate@active@char#1{%
1523     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1524   \let\bbl@s@switch@sh\bbl@switch@sh
1525   \def\bbl@switch@sh#1#2{%
1526     \ifx#2\@nnil\else
1527     \bbl@afterfi
1528     \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1529     \fi}
1530   \let\bbl@s@activate\bbl@activate
1531   \def\bbl@activate#1{%
1532     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1533   \let\bbl@s@deactivate\bbl@deactivate
1534   \def\bbl@deactivate#1{%
1535     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1536 \fi

```

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

or off.

```
1537 \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.

```
1538 \def\bbl@prim@s{%
1539   \prime\futurelet\@let@token\bbl@pr@m@s}
1540 \def\bbl@if@primes#1#2{%
1541   \ifx#1\@let@token
1542     \expandafter\@firstoftwo
1543   \else\ifx#2\@let@token
1544     \bbl@afterelse\expandafter\@firstoftwo
1545   \else
1546     \bbl@afterfi\expandafter\@secondoftwo
1547   \fi\fi}
1548 \begingroup
1549   \catcode`\^=7 \catcode`\*=active \lccode`\^=\^
1550   \catcode`\'=12 \catcode`\ "=active \lccode`\ "=\ '
1551   \lowercase{%
1552     \gdef\bbl@pr@m@s{%
1553       \bbl@if@primes" '%
1554       \pr@@@s
1555       {\bbl@if@primes*\pr@@@t\egroup}}
1556 \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).

```
1557 \initiate@active@char{~}
1558 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1559 \bbl@activate{~}
```

`\OT1dqpos` The position of the double quote character is different for the OT1 and T1 encodings. It will later be
`\T1dqpos` selected using the `\f@encoding` macro. Therefore we define two macros here to store the position of the character in these encodings.

```
1560 \expandafter\def\csname OT1dqpos\endcsname{127}
1561 \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

```
1562 \ifx\f@encoding\undefined
1563   \def\f@encoding{OT1}
1564 \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.

```
1565 \bbl@trace{Language attributes}
1566 \newcommand\languageattribute[2]{%
1567   \def\bbl@tempc{#1}%
1568   \bbl@fixname\bbl@tempc
1569   \bbl@iflanguage\bbl@tempc{%
1570     \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@attrs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```

1571 \ifx\bbl@known@attrs\undefined
1572 \in@false
1573 \else
1574 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
1575 \fi
1576 \ifin@
1577 \bbl@warning{%
1578     You have more than once selected the attribute '##1'\%
1579     for language #1. Reported}%
1580 \else

```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX -code.

```

1581 \bbl@exp{%
1582     \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
1583 \edef\bbl@tempa{\bbl@tempc-##1}%
1584 \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1585 {\csname\bbl@tempc @attr@##1\endcsname}%
1586 {\@attrerr{\bbl@tempc}{##1}}%
1587 \fi}}
1588 \@onlypreamble\languageattribute

```

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

```

1589 \newcommand*{\@attrerr}[2]{%
1590     \bbl@error
1591     {The attribute #2 is unknown for language #1.}%
1592     {Your command will be ignored, type <return> to proceed}}

```

`\bbl@declare@attribute` 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}`.

```

1593 \def\bbl@declare@attribute#1#2#3{%
1594     \bbl@xin@{,#2,}{,\BabelModifiers,}%
1595     \ifin@
1596         \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1597     \fi
1598     \bbl@add@list\bbl@attributes{#1-#2}%
1599     \expandafter\def\csname#1@attr@#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret \TeX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded. The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```

1600 \def\bbl@ifattributeset#1#2#3#4{%
1601     \ifx\bbl@known@attrs\undefined
1602     \in@false
1603     \else
1604         \bbl@xin@{,#1-#2,}{,\bbl@known@attrs,}%
1605     \fi
1606     \ifin@
1607         \bbl@afterelse#3%
1608     \else
1609         \bbl@afterfi#4%
1610     \fi}

```

`\bbl@ifknown@ttrib` An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX -code to be executed when the attribute is known and the \TeX -code to be executed otherwise.

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```

1611 \def\bbl@ifknown@ttrib#1#2{%
1612   \let\bbl@tempa\@secondoftwo
1613   \bbl@loopx\bbl@tempb{#2}{%
1614     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1615     \ifin@
1616       \let\bbl@tempa\@firstoftwo
1617     \else
1618       \fi}%
1619   \bbl@tempa}

```

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

```

1620 \def\bbl@clear@ttribs{%
1621   \ifx\bbl@attributes\undefined\else
1622     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1623       \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1624     \let\bbl@attributes\undefined
1625   \fi}
1626 \def\bbl@clear@ttrib#1-#2.{%
1627   \expandafter\let\csname#1@attr#2\endcsname\undefined}
1628 \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`

```

1629 \bbl@trace{Macros for saving definitions}
1630 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1631 \newcount\babel@savecnt
1632 \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.

```

1633 \def\babel@save#1{%
1634   \def\bbl@tempa{, #1,}% Clumsy, for Plain
1635   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1636     \expandafter{\expandafter,\bbl@savedextras,}}%
1637   \expandafter\in@\bbl@tempa
1638   \ifin@\else
1639     \bbl@add\bbl@savedextras{, #1,}%
1640     \bbl@carg\let\babel@number\babel@savecnt\#1\relax
1641     \toks@\expandafter{\originalTeX\let#1=}
1642     \bbl@exp{%
1643       \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1644     \advance\babel@savecnt\@ne

```

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

```

1645 \fi}
1646 \def\babel@savevariable#1{%
1647 \toks@\expandafter{\originalTeX #1}%
1648 \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 auxiliary macro is defined, but the main part is in `\babelprovide`. This new method should be ideally the default one.

```

1649 \def\bbl@frenchspacing{%
1650 \ifnum\the\scode`\.=\@m
1651 \let\bbl@nonfrenchspacing\relax
1652 \else
1653 \frenchspacing
1654 \let\bbl@nonfrenchspacing\nonfrenchspacing
1655 \fi}
1656 \let\bbl@nonfrenchspacing\nonfrenchspacing
1657 \let\bbl@elt\relax
1658 \edef\bbl@fs@chars{%
1659 \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1660 \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1661 \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1662 \def\bbl@pre@fs{%
1663 \def\bbl@elt##1##2##3{\scode`##1=\the\scode`##1\relax}%
1664 \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1665 \def\bbl@post@fs{%
1666 \bbl@save@sfcodes
1667 \edef\bbl@tempa{\bbl@cl{frspc}}}%
1668 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1669 \if u\bbl@tempa % do nothing
1670 \else\if n\bbl@tempa % non french
1671 \def\bbl@elt##1##2##3{%
1672 \ifnum\scode`##1=##2\relax
1673 \babel@savevariable{\scode`##1}%
1674 \scode`##1=##3\relax
1675 \fi}%
1676 \bbl@fs@chars
1677 \else\if y\bbl@tempa % french
1678 \def\bbl@elt##1##2##3{%
1679 \ifnum\scode`##1=##3\relax
1680 \babel@savevariable{\scode`##1}%
1681 \scode`##1=##2\relax
1682 \fi}%
1683 \bbl@fs@chars
1684 \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.

```

1685 \bbl@trace{Short tags}
1686 \def\babeltags#1{%
1687 \edef\bbl@tempa{\zap@space#1 \@empty}%
1688 \def\bbl@tempb##1=##2\@{ }%
1689 \edef\bbl@tempc{%
1690 \noexpand\newcommand
1691 \expandafter\noexpand\csname ##1\endcsname{%
1692 \noexpand\protect
1693 \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1694 \noexpand\newcommand

```

```

1695 \expandafter\noexpand\csname text##1\endcsname{%
1696 \noexpand\foreignlanguage{##2}}}%
1697 \bbl@tempc}%
1698 \bbl@for\bbl@tempa\bbl@tempa{%
1699 \expandafter\bbl@tempb\bbl@tempa\@@}}

```

4.9 Hyphens

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

```

1700 \bbl@trace{Hyphens}
1701 \onlypreamble\babelhyphenation
1702 \AtEndOfPackage{%
1703 \newcommand\babelhyphenation[2][\@empty]{%
1704 \ifx\bbl@hyphenation@relax
1705 \let\bbl@hyphenation@\@empty
1706 \fi
1707 \ifx\bbl@hyphlist\@empty\else
1708 \bbl@warning{%
1709 You must not intermingle \string\selectlanguage\space and\\%
1710 \string\babelhyphenation\space or some exceptions will not\\%
1711 be taken into account. Reported}%
1712 \fi
1713 \ifx\@empty#1%
1714 \protected@edef\bbl@hyphenation@\bbl@hyphenation@\space#2}%
1715 \else
1716 \bbl@vforeach{#1}{%
1717 \def\bbl@tempa{##1}%
1718 \bbl@fixname\bbl@tempa
1719 \bbl@iflanguage\bbl@tempa{%
1720 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1721 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1722 {}%
1723 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1724 #2}}}%
1725 \fi}}

```

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

```

1726 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1727 \def\bbl@t@one{T1}
1728 \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`.

```

1729 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1730 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1731 \def\bbl@hyphen{%
1732 \ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1733 \def\bbl@hyphen@i#1#2{%
1734 \bbl@ifunset{bbl@hy#1#2\@empty}%
1735 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1736 {\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.

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

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.

```
1737 \def\bbl@usehyphen#1{%
1738   \leavevmode
1739   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1740   \nobreak\hskip\z@skip}
1741 \def\bbl@usehyphen#1{%
1742   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1743 \def\bbl@hyphenchar{%
1744   \ifnum\hyphenchar\font=\m@ne
1745     \babe\nullhyphen
1746   \else
1747     \char\hyphenchar\font
1748   \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.

```
1749 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1750 \def\bbl@hy@@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1751 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1752 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1753 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1754 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1755 \def\bbl@hy@repeat{%
1756   \bbl@usehyphen{%
1757     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1758 \def\bbl@hy@repeat{%
1759   \bbl@usehyphen{%
1760     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1761 \def\bbl@hy@empty{\hskip\z@skip}
1762 \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.

```
1763 \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.

```
1764 \bbl@trace{Multiencoding strings}
1765 \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).

```
1766 \ifpackagewith{babel}{nocase}%
1767   {\let\bbl@patchuclc\relax}%
```

```

1768 {\def\bbbl@patchucllc{% TODO. Delete. Doesn't work any more.
1769 \global\let\bbbl@patchucllc\relax
1770 \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbbl@ucllc}}%
1771 \gdef\bbbl@ucllc##1{%
1772 \let\bbbl@encoded\bbbl@encoded@ucllc
1773 \bbbl@ifunset{\language @bbbl@ucllc}% and resumes it
1774 {##1}%
1775 {\let\bbbl@tempa##1\relax % Used by LANG@bbbl@ucllc
1776 \csname\language @bbbl@ucllc\endcsname}%
1777 {\bbbl@tolower\@empty}{\bbbl@toupper\@empty}}%
1778 \gdef\bbbl@tolower{\csname\language @bbbl@lc\endcsname}%
1779 \gdef\bbbl@toupper{\csname\language @bbbl@uc\endcsname}}%

1780 <<(*More package options)>> ≡
1781 \DeclareOption{nocase}{}
1782 <</More package options>>

```

The following package options control the behavior of \SetString.

```

1783 <<(*More package options)>> ≡
1784 \let\bbbl@opt@strings\@nnil % accept strings=value
1785 \DeclareOption{strings}{\def\bbbl@opt@strings{\BabelStringsDefault}}
1786 \DeclareOption{strings=encoded}{\let\bbbl@opt@strings\relax}
1787 \def\BabelStringsDefault{generic}
1788 <</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.

```

1789 \@onlypreamble\StartBabelCommands
1790 \def\StartBabelCommands{%
1791 \begingroup
1792 \@tempcnta="7F
1793 \def\bbbl@tempa{%
1794 \ifnum\@tempcnta>"FF\else
1795 \catcode\@tempcnta=11
1796 \advance\@tempcnta\@ne
1797 \expandafter\bbbl@tempa
1798 \fi}%
1799 \bbbl@tempa
1800 <<Macros local to BabelCommands>>
1801 \def\bbbl@provstring##1##2{%
1802 \providecommand##1{##2}%
1803 \bbbl@tglobal##1}%
1804 \global\let\bbbl@scafter\@empty
1805 \let\StartBabelCommands\bbbl@startcmds
1806 \ifx\BabelLanguages\relax
1807 \let\BabelLanguages\CurrentOption
1808 \fi
1809 \begingroup
1810 \let\bbbl@screaset\@nnil % local flag - disable 1st stopcommands
1811 \StartBabelCommands}
1812 \def\bbbl@startcmds{%
1813 \ifx\bbbl@screaset\@nnil\else
1814 \bbbl@usehooks{stopcommands}{}%
1815 \fi
1816 \endgroup
1817 \begingroup
1818 \@ifstar
1819 {\ifx\bbbl@opt@strings\@nnil
1820 \let\bbbl@opt@strings\BabelStringsDefault
1821 \fi
1822 \bbbl@startcmds@i}%
1823 \bbbl@startcmds@i}

```

```

1824 \def\bbl@startcmds@i#1#2{%
1825   \edef\bbl@L{\zap@space#1 \@empty}%
1826   \edef\bbl@G{\zap@space#2 \@empty}%
1827   \bbl@startcmds@ii}
1828 \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.

```

1829 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1830   \let\SetString\@gobbletwo
1831   \let\bbl@stringdef\@gobbletwo
1832   \let\AfterBabelCommands\@gobble
1833   \ifx\@empty#1%
1834     \def\bbl@sc@label{generic}%
1835     \def\bbl@encstring##1##2{%
1836       \ProvideTextCommandDefault##1{##2}%
1837       \bbl@toglobal##1%
1838       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1839     \let\bbl@sctest\in@true
1840   \else
1841     \let\bbl@sc@charset\space % <- zapped below
1842     \let\bbl@sc@fontenc\space % <- " "
1843     \def\bbl@tempa##1=##2\@nil{%
1844       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1845     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1846     \def\bbl@tempa##1 ##2{% space -> comma
1847       ##1%
1848       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1849     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1850     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1851     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1852     \def\bbl@encstring##1##2{%
1853       \bbl@foreach\bbl@sc@fontenc{%
1854         \bbl@ifunset{T@###1}%
1855         {}%
1856         {\ProvideTextCommand##1{###1}{##2}%
1857          \bbl@toglobal##1%
1858          \expandafter
1859          \bbl@toglobal\csname####1\string##1\endcsname}}}%
1860     \def\bbl@sctest{%
1861       \bbl@xin@{\bbl@opt@strings,}{\bbl@sc@label,\bbl@sc@fontenc,}}%
1862   \fi
1863   \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
1864   \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
1865     \let\AfterBabelCommands\bbl@aftercmds
1866     \let\SetString\bbl@setstring
1867     \let\bbl@stringdef\bbl@encstring
1868   \else % ie, strings=value
1869     \bbl@sctest
1870   \ifin@
1871     \let\AfterBabelCommands\bbl@aftercmds
1872     \let\SetString\bbl@setstring
1873     \let\bbl@stringdef\bbl@provstring
1874   \fi\fi\fi
1875   \bbl@scswitch
1876   \ifx\bbl@G\@empty

```

```

1877 \def\SetString##1##2{%
1878 \bbl@error{Missing group for string \string##1}%
1879 {You must assign strings to some category, typically\\%
1880 captions or extras, but you set none}}%
1881 \fi
1882 \ifx\@empty#1%
1883 \bbl@usehooks{defaultcommands}{}%
1884 \else
1885 \@expandtwoargs
1886 \bbl@usehooks{encodedcommands}{\bbl@sc@charset}\bbl@sc@fontenc}}%
1887 \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).

```

1888 \def\bbl@forlang#1#2{%
1889 \bbl@for#1\bbl@L{%
1890 \bbl@xin@{, #1, }{, \BabelLanguages,}%
1891 \ifin@#2\relax\fi}}
1892 \def\bbl@scswitch{%
1893 \bbl@forlang\bbl@tempa{%
1894 \ifx\bbl@G\@empty\else
1895 \ifx\SetString\@gobbleset\else
1896 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1897 \bbl@xin@{, \bbl@GL, }{, \bbl@screset,}%
1898 \ifin@\else
1899 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1900 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1901 \fi
1902 \fi
1903 \fi}}
1904 \AtEndOfPackage{%
1905 \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{#2}}}%
1906 \let\bbl@scswitch\relax}
1907 \onlypreamble\EndBabelCommands
1908 \def\EndBabelCommands{%
1909 \bbl@usehooks{stopcommands}{}%
1910 \endgroup
1911 \endgroup
1912 \bbl@scafter}
1913 \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.

```

1914 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1915 \bbl@forlang\bbl@tempa{%
1916 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1917 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1918 {\bbl@exp{%
1919 \global\\bbl@add\<\bbl@G\bbl@tempa>\\bbl@scset\\#1\<\bbl@LC>}}}%
1920 {}}%
1921 \def\BabelString{#2}%
1922 \bbl@usehooks{stringprocess}{}%

```

```

1923 \expandafter\bbL@stringdef
1924 \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`.

```

1925 \ifx\bbL@opt@strings\relax
1926 \def\bbL@scset#1#2{\def#1{\bbL@encoded#2}}
1927 \bbL@patchuclC
1928 \let\bbL@encoded\relax
1929 \def\bbL@encoded@uclC#1{%
1930 \inmathwarn#1%
1931 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1932 \expandafter\ifx\csname ?\string#1\endcsname\relax
1933 \TextSymbolUnavailable#1%
1934 \else
1935 \csname ?\string#1\endcsname
1936 \fi
1937 \else
1938 \csname\cf@encoding\string#1\endcsname
1939 \fi}
1940 \else
1941 \def\bbL@scset#1#2{\def#1{#2}}
1942 \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.

```

1943 <<*Macros local to BabelCommands>> ≡
1944 \def\SetStringLoop##1##2{%
1945 \def\bbL@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1946 \count@\z@
1947 \bbL@loop\bbL@tempa{##2}{% empty items and spaces are ok
1948 \advance\count@\@ne
1949 \toks@\expandafter{\bbL@tempa}%
1950 \bbL@exp{%
1951 \\\SetString\bbL@templ{\romannumeral\count@}{\the\toks@}%
1952 \count@=\the\count@\relax}}%
1953 <</Macros local to BabelCommands>>

```

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

```

1954 \def\bbL@aftercmds#1{%
1955 \toks@\expandafter{\bbL@scafter#1}%
1956 \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*.

```

1957 <<*Macros local to BabelCommands>> ≡
1958 \newcommand\SetCase[3][{}]{%
1959 \bbL@patchuclC
1960 \bbL@forlang\bbL@tempa{%
1961 \bbL@carg\bbL@encstring{\bbL@tempa @bbL@uclC}{\bbL@tempa##1}%
1962 \bbL@carg\bbL@encstring{\bbL@tempa @bbL@uc}{##2}%
1963 \bbL@carg\bbL@encstring{\bbL@tempa @bbL@lc}{##3}}}%
1964 <</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.

```

1965 <<*Macros local to BabelCommands>> ≡
1966 \newcommand\SetHyphenMap[1]{%

```



```

1967 \bbl@forlang\bbl@tempa{%
1968 \expandafter\bbl@stringdef
1969 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1970 <</Macros local to BabelCommands>>

```

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

```

1971 \newcommand\BabelLower[2]{% one to one.
1972 \ifnum\lccode#1=#2\else
1973 \babel@savevariable{\lccode#1}%
1974 \lccode#1=#2\relax
1975 \fi}
1976 \newcommand\BabelLowerMM[4]{% many-to-many
1977 \@tempcnta=#1\relax
1978 \@tempcntb=#4\relax
1979 \def\bbl@tempa{%
1980 \ifnum\@tempcnta>#2\else
1981 \expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1982 \advance\@tempcnta#3\relax
1983 \advance\@tempcntb#3\relax
1984 \expandafter\bbl@tempa
1985 \fi}%
1986 \bbl@tempa}
1987 \newcommand\BabelLowerM0[4]{% many-to-one
1988 \@tempcnta=#1\relax
1989 \def\bbl@tempa{%
1990 \ifnum\@tempcnta>#2\else
1991 \expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1992 \advance\@tempcnta#3
1993 \expandafter\bbl@tempa
1994 \fi}%
1995 \bbl@tempa}

```

The following package options control the behavior of hyphenation mapping.

```

1996 <<(*More package options)>> ≡
1997 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1998 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1999 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2000 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2001 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2002 <</More package options>>

```

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

```

2003 \AtEndOfPackage{%
2004 \ifx\bbl@opt@hyphenmap\undefined
2005 \bbl@xin@{,}{\bbl@language@opts}%
2006 \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2007 \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.

```

2008 \newcommand\setlocalecaption{% TODO. Catch typos.
2009 \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2010 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2011 \bbl@trim@def\bbl@tempa{#2}%
2012 \bbl@xin@{.template}{\bbl@tempa}%
2013 \ifin@
2014 \bbl@ini@captions@template{#3}{#1}%
2015 \else
2016 \edef\bbl@tempd{%
2017 \expandafter\expandafter\expandafter
2018 \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2019 \bbl@xin@
2020 {\expandafter\string\csname #2name\endcsname}%

```

```

2021     {\bbl@tempd}%
2022 \ifin@ % Renew caption
2023     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2024     \ifin@
2025         \bbl@exp{%
2026             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2027             {\bbl@scset\<#2name>\<#1#2name>}%
2028             }%
2029 \else % Old way converts to new way
2030     \bbl@ifunset{#1#2name}%
2031         {\bbl@exp{%
2032             \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2033             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2034             {\def\<#2name>{\<#1#2name>}}%
2035             }%
2036         }%
2037     \fi
2038 \else
2039     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2040     \ifin@ % New way
2041         \bbl@exp{%
2042             \\bbl@add\<captions#1>{\bbl@scset\<#2name>\<#1#2name>}%
2043             \\bbl@ifsamestring{\bbl@tempa}{\language}%
2044             {\bbl@scset\<#2name>\<#1#2name>}%
2045             }%
2046 \else % Old way, but defined in the new way
2047     \bbl@exp{%
2048         \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2049         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2050         {\def\<#2name>{\<#1#2name>}}%
2051         }%
2052     \fi%
2053 \fi
2054 \@namedef{#1#2name}{#3}%
2055 \toks@{\expandafter{\bbl@captionslist}}%
2056 \bbl@exp{\in@{\<#2name>}{\the\toks@}}%
2057 \ifin@\else
2058     \bbl@exp{\bbl@add\bbl@captionslist{\<#2name>}}%
2059     \bbl@toglobal\bbl@captionslist
2060 \fi
2061 \fi}
2062 % \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.

```

2063 \bbl@trace{Macros related to glyphs}
2064 \def\set@low@box#1{\setbox\tw\hbox{,}\setbox\z@ \hbox{#1}%
2065     \dimen\z@ \ht\z@ \advance\dimen\z@ -\ht\tw@%
2066     \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.

```

2067 \def\save@s f@q#1{\leavevmode
2068     \begingroup
2069     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2070     \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.

```
2071 \ProvideTextCommand{\quotedblbase}{OT1}{%
2072   \save@sf@q{\set@low@box{\textquotedblright\}}%
2073   \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.

```
2074 \ProvideTextCommandDefault{\quotedblbase}{%
2075   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2076 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2077   \save@sf@q{\set@low@box{\textquoteright\}}%
2078   \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.

```
2079 \ProvideTextCommandDefault{\quotesinglbase}{%
2080   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2081 \ProvideTextCommand{\guillemetleft}{OT1}{%
2082   \ifmmode
2083     \ll
2084   \else
2085     \save@sf@q{\nobreak
2086       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2087     \fi}
2088 \ProvideTextCommand{\guillemetright}{OT1}{%
2089   \ifmmode
2090     \gg
2091   \else
2092     \save@sf@q{\nobreak
2093       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2094     \fi}
2095 \ProvideTextCommand{\guillemotleft}{OT1}{%
2096   \ifmmode
2097     \ll
2098   \else
2099     \save@sf@q{\nobreak
2100       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2101     \fi}
2102 \ProvideTextCommand{\guillemotright}{OT1}{%
2103   \ifmmode
2104     \gg
2105   \else
2106     \save@sf@q{\nobreak
2107       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2108     \fi}
```

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

```
2109 \ProvideTextCommandDefault{\guillemetleft}{%
2110   \UseTextSymbol{OT1}{\guillemetleft}}
2111 \ProvideTextCommandDefault{\guillemetright}{%
2112   \UseTextSymbol{OT1}{\guillemetright}}
2113 \ProvideTextCommandDefault{\guillemotleft}{%
2114   \UseTextSymbol{OT1}{\guillemotleft}}
2115 \ProvideTextCommandDefault{\guillemotright}{%
2116   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```

2117 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2118   \ifmmode
2119     <%
2120   \else
2121     \save@sf@q{\nobreak
2122       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2123   \fi}
2124 \ProvideTextCommand{\guilsinglright}{OT1}{%
2125   \ifmmode
2126     >%
2127   \else
2128     \save@sf@q{\nobreak
2129       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2130   \fi}

```

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

```

2131 \ProvideTextCommandDefault{\guilsinglleft}{%
2132   \UseTextSymbol{OT1}{\guilsinglleft}}
2133 \ProvideTextCommandDefault{\guilsinglright}{%
2134   \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.

```

2135 \DeclareTextCommand{\ij}{OT1}{%
2136   i\kern-0.02em\bbl@allowhyphens j}
2137 \DeclareTextCommand{\IJ}{OT1}{%
2138   I\kern-0.02em\bbl@allowhyphens J}
2139 \DeclareTextCommand{\ij}{T1}{\char188}
2140 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2141 \ProvideTextCommandDefault{\ij}{%
2142   \UseTextSymbol{OT1}{\ij}}
2143 \ProvideTextCommandDefault{\IJ}{%
2144   \UseTextSymbol{OT1}{\IJ}}

```

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

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

```

2145 \def\crrtic@{\hrule height0.1ex width0.3em}
2146 \def\crttic@{\hrule height0.1ex width0.33em}
2147 \def\ddj@{%
2148   \setbox0\hbox{d}\dimen@=\ht0
2149   \advance\dimen@lex
2150   \dimen@.45\dimen@
2151   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2152   \advance\dimen@ii.5ex
2153   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2154 \def\DDJ@{%
2155   \setbox0\hbox{D}\dimen@=.55\ht0
2156   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2157   \advance\dimen@ii.15ex % correction for the dash position
2158   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2159   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2160   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2161 %
2162 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2163 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```
2164 \ProvideTextCommandDefault{\dj}{%
2165   \UseTextSymbol{OT1}{\dj}}
2166 \ProvideTextCommandDefault{\DJ}{%
2167   \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.

```
2168 \DeclareTextCommand{\SS}{OT1}{SS}
2169 \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
2170 \ProvideTextCommandDefault{\glq}{%
2171   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2172 \ProvideTextCommand{\grq}{T1}{%
2173   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}%
2174 \ProvideTextCommand{\grq}{TU}{%
2175   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2176 \ProvideTextCommand{\grq}{OT1}{%
2177   \save@sf@q{\kern-.0125em
2178     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2179     \kern.07em\relax}}
2180 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The ‘german’ double quotes.

```
\grqq
2181 \ProvideTextCommandDefault{\glqq}{%
2182   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2183 \ProvideTextCommand{\grqq}{T1}{%
2184   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2185 \ProvideTextCommand{\grqq}{TU}{%
2186   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2187 \ProvideTextCommand{\grqq}{OT1}{%
2188   \save@sf@q{\kern-.07em
2189     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2190     \kern.07em\relax}}
2191 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq The ‘french’ single guillemets.

```
\frq
2192 \ProvideTextCommandDefault{\flq}{%
2193   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}%
2194 \ProvideTextCommandDefault{\frq}{%
2195   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq The ‘french’ double guillemets.

```
\frqq
2196 \ProvideTextCommandDefault{\flqq}{%
2197   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}%
2198 \ProvideTextCommandDefault{\frqq}{%
2199   \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).

```
2200 \def\umlauthigh{%
2201   \def\bbl@umlauta##1{\leavevmode\bgroup%
2202     \accent\csname\fontencoding dqpos\endcsname
2203     ##1\bbl@allowhyphens\egroup}%
2204   \let\bbl@umlaute\bbl@umlauta}
2205 \def\umlautlow{%
2206   \def\bbl@umlauta{\protect\lower@umlaut}}
2207 \def\umlautelower{%
2208   \def\bbl@umlaute{\protect\lower@umlaut}}
2209 \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.

```
2210 \expandafter\ifx\csname U@D\endcsname\relax
2211   \csname newdimen\endcsname\U@D
2212 \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.

```
2213 \def\lower@umlaut#1{%
2214   \leavevmode\bgroup
2215   \U@D lex%
2216   {\setbox\z@\hbox{%
2217     \char\csname\fontencoding dqpos\endcsname}%
2218     \dimen@ -.45ex\advance\dimen@\ht\z@
2219     \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2220   \accent\csname\fontencoding dqpos\endcsname
2221   \fontdimen5\font\U@D #1%
2222   \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).

```
2223 \AtBeginDocument{%
2224   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2225   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2226   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2227   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2228   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2229   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2230   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2231   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2232   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2233   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2234   \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.

```

2235 \ifx\l@english\@undefined
2236   \chardef\l@english\z@
2237 \fi
2238 % The following is used to cancel rules in ini files (see Amharic).
2239 \ifx\l@unhyphenated\@undefined
2240   \newlanguage\l@unhyphenated
2241 \fi

```

4.13 Layout

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

```

2242 \bbl@trace{Bidi layout}
2243 \providecommand\IfBabelLayout[3]{#3}%
2244 <-core>
2245 \newcommand\BabelPatchSection[1]{%
2246   \@ifundefined{#1}{}{%
2247     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2248     \@namedef{#1}{%
2249       \ifstar{\bbl@presec@#1}%
2250       {\@dblarg{\bbl@presec@#1}}}%
2251 \def\bbl@presec@#1[#2]#3{%
2252   \bbl@exp{%
2253     \\\select@language@x{\bbl@main@language}%
2254     \\\bbl@cs{sspre@#1}%
2255     \\\bbl@cs{ss@#1}%
2256     [\\foreignlanguage{\language}{\unexpanded{#2}}}%
2257     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2258     \\\select@language@x{\language}}%
2259 \def\bbl@presec@#1#2{%
2260   \bbl@exp{%
2261     \\\select@language@x{\bbl@main@language}%
2262     \\\bbl@cs{sspre@#1}%
2263     \\\bbl@cs{ss@#1}*%
2264     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2265     \\\select@language@x{\language}}%
2266 \IfBabelLayout{sectioning}%
2267   {\BabelPatchSection{part}%
2268    \BabelPatchSection{chapter}%
2269    \BabelPatchSection{section}%
2270    \BabelPatchSection{subsection}%
2271    \BabelPatchSection{subsubsection}%
2272    \BabelPatchSection{paragraph}%
2273    \BabelPatchSection{subparagraph}%
2274    \def\babel@toc#1{%
2275      \select@language@x{\bbl@main@language}}}%
2276 \IfBabelLayout{captions}%
2277   {\BabelPatchSection{caption}}}%
2278 <+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.

```

2279 \bbl@trace{Input engine specific macros}
2280 \ifcase\bbl@engine
2281   \input txtbabel.def
2282 \or
2283   \input luababel.def
2284 \or
2285   \input xebabel.def

```

```

2286 \fi
2287 \providecommand\babelfont{%
2288   \bbl@error
2289   {This macro is available only in LuaLaTeX and XeLaTeX.}%
2290   {Consider switching to these engines.}}
2291 \providecommand\babelprehyphenation{%
2292   \bbl@error
2293   {This macro is available only in LuaLaTeX.}%
2294   {Consider switching to that engine.}}
2295 \ifx\babelposthyphenation\@undefined
2296   \let\babelposthyphenation\babelprehyphenation
2297   \let\babelpatterns\babelprehyphenation
2298   \let\babelcharproperty\babelprehyphenation
2299 \fi

```

4.15 Creating and modifying languages

Continue with \TeX 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.

```

2300 </package | core>
2301 <*package>
2302 \bbl@trace{Creating languages and reading ini files}
2303 \let\bbl@extend@ini\@gobble
2304 \newcommand\babelprovide[2][]{%
2305   \let\bbl@savelangname\language
2306   \edef\bbl@savelocaleid{\the\localeid}%
2307   % Set name and locale id
2308   \edef\language{#2}%
2309   \bbl@id@assign
2310   % Initialize keys
2311   \bbl@vforeach{captions,date,import,main,script,language,%
2312     hyphenrules,linebreaking,justification,mapfont,maparabic,%
2313     mapdigits,intraspaces,intrapenalty,onchar,transforms,alpha,%
2314     Alph,labels,labels*,calendar,date,casing}%
2315     {\bbl@csarg\let{KVP@##1}\@nnil}%
2316   \global\let\bbl@release@transforms\@empty
2317   \let\bbl@calendars\@empty
2318   \global\let\bbl@inidata\@empty
2319   \global\let\bbl@extend@ini\@gobble
2320   \global\let\bbl@included@inis\@empty
2321   \gdef\bbl@key@list{;}%
2322   \bbl@forkv{#1}{%
2323     \in@{/}{##1}% With /, (re)sets a value in the ini
2324     \ifin@
2325       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2326       \bbl@renewinikey##1\@{##2}%
2327     \else
2328       \bbl@csarg\ifx{KVP@##1}\@nnil\else
2329         \bbl@error
2330         {Unknown key '##1' in \string\babelprovide}%
2331         {See the manual for valid keys}%
2332       \fi
2333       \bbl@csarg\def{KVP@##1}{##2}%
2334     \fi}%
2335   \chardef\bbl@howloaded=0:none; 1:ldf without ini; 2:ini
2336   \bbl@ifunset{date#2}\z@{\bbl@ifunset\bbl@llevel@#2}\@ne\tw@}%
2337   % == init ==
2338   \ifx\bbl@screset\@undefined
2339     \bbl@ldfinit
2340   \fi
2341   % == date (as option) ==

```



```

2342 % \ifx\bbbl@KVP@date\@nnil\else
2343 % \fi
2344 % ==
2345 \let\bbbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2346 \ifcase\bbbl@howloaded
2347   \let\bbbl@lbkflag\@empty % new
2348 \else
2349   \ifx\bbbl@KVP@hyphenrules\@nnil\else
2350     \let\bbbl@lbkflag\@empty
2351   \fi
2352   \ifx\bbbl@KVP@import\@nnil\else
2353     \let\bbbl@lbkflag\@empty
2354   \fi
2355 \fi
2356 % == import, captions ==
2357 \ifx\bbbl@KVP@import\@nnil\else
2358   \bbl@exp{\bbbl@ifblank{\bbbl@KVP@import}}%
2359   {\ifx\bbbl@initoload\relax
2360     \begingroup
2361       \def\BabelBeforeIni##1##2{\gdef\bbbl@KVP@import{##1}\endinput}%
2362       \bbl@input@texini{#2}%
2363     \endgroup
2364   \else
2365     \xdef\bbbl@KVP@import{\bbbl@initoload}%
2366   \fi}%
2367 {}%
2368 \let\bbbl@KVP@date\@empty
2369 \fi
2370 \let\bbbl@KVP@captions@@\bbbl@KVP@captions % TODO. A dirty hack
2371 \ifx\bbbl@KVP@captions\@nnil
2372   \let\bbbl@KVP@captions\bbbl@KVP@import
2373 \fi
2374 % ==
2375 \ifx\bbbl@KVP@transforms\@nnil\else
2376   \bbl@replace\bbbl@KVP@transforms{ }{,}%
2377 \fi
2378 % == Load ini ==
2379 \ifcase\bbbl@howloaded
2380   \bbl@provide@new{#2}%
2381 \else
2382   \bbl@ifblank{#1}%
2383   {}% With \bbl@load@basic below
2384   {\bbl@provide@renew{#2}}%
2385 \fi
2386 % == include == TODO
2387 % \ifx\bbbl@included@inis\@empty\else
2388 %   \bbl@replace\bbbl@included@inis{ }{,}%
2389 %   \bbl@foreach\bbbl@included@inis{%
2390 %     \openin\bbbl@readstream=babel-##1.ini
2391 %     \bbl@extend@ini{#2}%
2392 %     \closein\bbbl@readstream
2393 %   \fi
2394 % Post tasks
2395 % -----
2396 % == subsequent calls after the first provide for a locale ==
2397 \ifx\bbbl@inidata\@empty\else
2398   \bbl@extend@ini{#2}%
2399 \fi
2400 % == ensure captions ==
2401 \ifx\bbbl@KVP@captions\@nnil\else
2402   \bbl@ifunset{\bbbl@extracaps@#2}%
2403   {\bbl@exp{\bbbl@babelensure[exclude=\\today]{#2}}}%
2404   {\bbl@exp{\bbbl@babelensure[exclude=\\today,

```

```

2405         include=\[bbl@extracaps@#2]]{#2}}%
2406 \bbl@ifunset{bbl@ensure@\language}\language}%
2407 {\bbl@exp{%
2408   \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
2409     \\\foreignlanguage{\language}%
2410     {###1}}}%
2411 }%
2412 \bbl@exp{%
2413   \\\bbl@tglobal\<bbl@ensure@\language>%
2414   \\\bbl@tglobal\<bbl@ensure@\language\space>%
2415 \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.

```

2416 \bbl@load@basic{#2}%
2417 % == script, language ==
2418 % Override the values from ini or defines them
2419 \ifx\bbl@KVP@script\@nnil\else
2420   \bbl@csarg\edef{sname#2}{\bbl@KVP@script}%
2421 \fi
2422 \ifx\bbl@KVP@language\@nnil\else
2423   \bbl@csarg\edef{lname#2}{\bbl@KVP@language}%
2424 \fi
2425 \ifcase\bbl@engine\or
2426   \bbl@ifunset{bbl@chrng@\language}{}%
2427   {\directlua{
2428     Babel.set_chranges_b('\bbl@cl{sbcpr}', '\bbl@cl{chrng}') }}%
2429 \fi
2430 % == onchar ==
2431 \ifx\bbl@KVP@onchar\@nnil\else
2432   \bbl@luahyphenate
2433   \bbl@exp{%
2434     \\\AddToHook{env/document/before}{\select@language{#2}{}}}%
2435   \directlua{
2436     if Babel.locale_mapped == nil then
2437       Babel.locale_mapped = true
2438       Babel.linebreaking.add_before(Babel.locale_map, 1)
2439       Babel.loc_to_scr = {}
2440       Babel.chr_to_loc = Babel.chr_to_loc or {}
2441     end
2442     Babel.locale_props[\the\localeid].letters = false
2443   }%
2444   \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2445   \ifin@
2446     \directlua{
2447       Babel.locale_props[\the\localeid].letters = true
2448     }%
2449   \fi
2450   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2451   \ifin@
2452     \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
2453       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2454     \fi
2455     \bbl@exp{\bbl@add\bbl@starthyphens
2456       {\bbl@patterns@lua{\language}}}%
2457     % TODO - error/warning if no script
2458     \directlua{
2459       if Babel.script_blocks['\bbl@cl{sbcpr}'] then
2460         Babel.loc_to_scr[\the\localeid] =
2461           Babel.script_blocks['\bbl@cl{sbcpr}']
2462         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2463         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language}\space

```

```

2464         end
2465     }%
2466 \fi
2467 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2468 \ifin@
2469     \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{}%
2470     \bbl@ifunset{bbl@wdir@\languageName}{\bbl@provide@dirs{\languageName}}{}%
2471     \directlua{
2472         if Babel.script_blocks['\bbl@cl{sbc}'] then
2473             Babel.loc_to_scr[\the\localeid] =
2474                 Babel.script_blocks['\bbl@cl{sbc}']
2475         end}%
2476 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
2477     \AtBeginDocument{%
2478         \bbl@patchfont{\bbl@mapselect}}%
2479         {\selectfont}}%
2480     \def\bbl@mapselect{%
2481         \let\bbl@mapselect\relax
2482         \edef\bbl@prefontid{\fontid\font}}%
2483     \def\bbl@mapdir##1{%
2484         {\def\languageName{##1}%
2485         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2486         \bbl@switchfont
2487         \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2488             \directlua{
2489                 Babel.locale_props[\the\csname bbl@id@##1\endcsname]%
2490                     ['\bbl@prefontid'] = \fontid\font\space}%
2491             \fi}}%
2492     \fi
2493     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languageName}}}%
2494 \fi
2495 % TODO - catch non-valid values
2496 \fi
2497 % == mapfont ==
2498 % For bidi texts, to switch the font based on direction
2499 \ifx\bbl@KVP@mapfont\@nnil\else
2500     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}{}%
2501     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2502         mapfont. Use 'direction'.%
2503         {See the manual for details.}}}%
2504     \bbl@ifunset{bbl@lsys@\languageName}{\bbl@provide@lsys{\languageName}}{}%
2505     \bbl@ifunset{bbl@wdir@\languageName}{\bbl@provide@dirs{\languageName}}{}%
2506 \ifx\bbl@mapselect\undefined % TODO. See onchar.
2507     \AtBeginDocument{%
2508         \bbl@patchfont{\bbl@mapselect}}%
2509         {\selectfont}}%
2510     \def\bbl@mapselect{%
2511         \let\bbl@mapselect\relax
2512         \edef\bbl@prefontid{\fontid\font}}%
2513     \def\bbl@mapdir##1{%
2514         {\def\languageName{##1}%
2515         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2516         \bbl@switchfont
2517         \directlua{Babel.fontmap
2518             [\the\csname bbl@wdir@##1\endcsname]%
2519             [\bbl@prefontid]=\fontid\font}}}%
2520     \fi
2521     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\languageName}}}%
2522 \fi
2523 % == Line breaking: intraspace, intrapenalty ==
2524 % For CJK, East Asian, Southeast Asian, if interspace in ini
2525 \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2526     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%

```

```

2527 \fi
2528 \bbl@provide@intraspace
2529 % == Line breaking: CJK quotes == TODO -> @extras
2530 \ifcase\bbl@engine\or
2531   \bbl@xin@{/c}{/\bbl@cl{\lnbrk}}%
2532   \ifin@
2533     \bbl@ifunset{\bbl@quote@\languagename}{}%
2534     {\directlua{
2535       Babel.locale_props[\the\localeid].cjk_quotes = {}
2536       local cs = 'op'
2537       for c in string.utfvalues(
2538         [[\csname bbl@quote@\languagename\endcsname]]) do
2539         if Babel.cjk_characters[c].c == 'qu' then
2540           Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2541         end
2542         cs = ( cs == 'op') and 'cl' or 'op'
2543       end
2544     }}%
2545   \fi
2546 \fi
2547 % == Line breaking: justification ==
2548 \ifx\bbl@KVP@justification\@nnil\else
2549   \let\bbl@KVP@linebreaking\bbl@KVP@justification
2550 \fi
2551 \ifx\bbl@KVP@linebreaking\@nnil\else
2552   \bbl@xin@{,\bbl@KVP@linebreaking,}%
2553   {,elongated,kashida,cjk,padding,unhyphenated,}%
2554   \ifin@
2555     \bbl@csarg\xdef
2556     {\lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2557   \fi
2558 \fi
2559 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
2560 \ifin@else\bbl@xin@{/k}{/\bbl@cl{\lnbrk}}\fi
2561 \ifin@\bbl@arabicjust\fi
2562 \bbl@xin@{/p}{/\bbl@cl{\lnbrk}}%
2563 \ifin@\AtBeginDocument{\@nameuse{\bbl@tibetanjust}}\fi
2564 % == Line breaking: hyphenate.other.(locale|script) ==
2565 \ifx\bbl@lbkflag\@empty
2566   \bbl@ifunset{\bbl@hyotl@\languagename}{}%
2567   {\bbl@csarg\bbl@replace{\hyotl@\languagename}{ }{,},}%
2568   \bbl@startcommands*\languagename}{}%
2569   \bbl@csarg\bbl@foreach{\hyotl@\languagename}{%
2570     \ifcase\bbl@engine
2571       \ifnum##1<257
2572         \SetHyphenMap{\BabelLower{##1}{##1}}%
2573       \fi
2574     \else
2575       \SetHyphenMap{\BabelLower{##1}{##1}}%
2576     \fi}%
2577   \bbl@endcommands}%
2578 \bbl@ifunset{\bbl@hyots@\languagename}{}%
2579 {\bbl@csarg\bbl@replace{\hyots@\languagename}{ }{,},}%
2580 \bbl@csarg\bbl@foreach{\hyots@\languagename}{%
2581   \ifcase\bbl@engine
2582     \ifnum##1<257
2583       \global\lccode##1=##1\relax
2584     \fi
2585   \else
2586     \global\lccode##1=##1\relax
2587   \fi}}%
2588 \fi
2589 % == Counters: maparabic ==

```

```

2590 % Native digits, if provided in ini (TeX level, xe and lua)
2591 \ifcase\bbbl@engine\else
2592   \bbbl@ifunset{\bbbl@dgnat@\language\name}{}%
2593   {\expandafter\ifx\csname \bbbl@dgnat@\language\name\endcsname\@empty\else
2594     \expandafter\expandafter\expandafter
2595     \bbbl@setdigits\csname \bbbl@dgnat@\language\name\endcsname
2596     \ifx\bbbl@KVP@maparabic\@nnil\else
2597       \ifx\bbbl@latinarabic\@undefined
2598         \expandafter\let\expandafter\@arabic
2599         \csname \bbbl@counter@\language\name\endcsname
2600       \else % ie, if layout=counters, which redefines \@arabic
2601         \expandafter\let\expandafter\bbbl@latinarabic
2602         \csname \bbbl@counter@\language\name\endcsname
2603       \fi
2604     \fi
2605   \fi}%
2606 \fi
2607 % == Counters: mapdigits ==
2608 % > luababel.def
2609 % == Counters: alph, Alph ==
2610 \ifx\bbbl@KVP@alph\@nnil\else
2611   \bbbl@exp{%
2612     \\bbbl@add\<\bbbl@preextras@\language\name>{%
2613       \\babel@save\\@alph
2614       \let\\@alph\<\bbbl@cntr@\bbbl@KVP@alph @\language\name>}}%
2615   \fi
2616 \ifx\bbbl@KVP@Alph\@nnil\else
2617   \bbbl@exp{%
2618     \\bbbl@add\<\bbbl@preextras@\language\name>{%
2619       \\babel@save\\@Alph
2620       \let\\@Alph\<\bbbl@cntr@\bbbl@KVP@Alph @\language\name>}}%
2621   \fi
2622 % == Casing ==
2623 \ifx\bbbl@KVP@casing\@nnil\else
2624   \bbbl@csarg\xdef{casing@\language\name}%
2625   {\@nameuse{\bbbl@casing@\language\name}-x-\bbbl@KVP@casing}%
2626 \fi
2627 % == Calendars ==
2628 \ifx\bbbl@KVP@calendar\@nnil
2629   \edef\bbbl@KVP@calendar{\bbbl@cl{calpr}}%
2630 \fi
2631 \def\bbbl@tempe##1 ##2\@{ % Get first calendar
2632   \def\bbbl@tempa{##1}}%
2633   \bbbl@exp{\\bbbl@tempe\bbbl@KVP@calendar\space\\@}%
2634 \def\bbbl@tempe##1.##2.##3\@{
2635   \def\bbbl@tempc{##1}%
2636   \def\bbbl@tempb{##2}}%
2637 \expandafter\bbbl@tempe\bbbl@tempa.\@
2638 \bbbl@csarg\xdef{calpr@\language\name}%
2639 \ifx\bbbl@tempc\@empty\else
2640   calendar=\bbbl@tempc
2641 \fi
2642 \ifx\bbbl@tempb\@empty\else
2643   ,variant=\bbbl@tempb
2644 \fi}%
2645 % == engine specific extensions ==
2646 % Defined in XXXbabel.def
2647 \bbbl@provide@extra{#2}%
2648 % == require.babel in ini ==
2649 % To load or reload the babel-*.tex, if require.babel in ini
2650 \ifx\bbbl@beforestart\relax\else % But not in doc aux or body
2651   \bbbl@ifunset{\bbbl@rqtex@\language\name}{}%
2652   {\expandafter\ifx\csname \bbbl@rqtex@\language\name\endcsname\@empty\else

```

```

2653 \let\BabelBeforeIni\@gobbletwo
2654 \chardef\atcatcode=\catcode`\@
2655 \catcode`\@=11\relax
2656 \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2657 \catcode`\@=\atcatcode
2658 \let\atcatcode\relax
2659 \global\bbl@csarg\let{rqtex@\language}\relax
2660 \fi}%
2661 \bbl@foreach\bbl@calendars{%
2662 \bbl@ifunset{\bbl@ca##1}{%
2663 \chardef\atcatcode=\catcode`\@
2664 \catcode`\@=11\relax
2665 \InputIfFileExists{babel-ca-##1.tex}{\fi}%
2666 \catcode`\@=\atcatcode
2667 \let\atcatcode\relax}%
2668 \fi}%
2669 \fi
2670 % == frenchspacing ==
2671 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2672 \ifin@ \else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2673 \ifin@
2674 \bbl@extras@wrap{\bbl@pre@fs}%
2675 {\bbl@pre@fs}%
2676 {\bbl@post@fs}%
2677 \fi
2678 % == transforms ==
2679 % > luababel.def
2680 % == main ==
2681 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2682 \let\language\bbl@savelangname
2683 \chardef\localeid\bbl@savelocaleid\relax
2684 \fi
2685 % == hyphenrules (apply if current) ==
2686 \ifx\bbl@KVP@hyphenrules\@nnil\else
2687 \ifnum\bbl@savelocaleid=\localeid
2688 \language\@nameuse{l@\language}%
2689 \fi
2690 \fi}

```

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

```

2691 \def\bbl@provide@new#1{%
2692 \namedef{date#1}{}}% marks lang exists - required by \StartBabelCommands
2693 \namedef{extras#1}{}%
2694 \namedef{noextras#1}{}%
2695 \bbl@startcommands*{#1}{captions}%
2696 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2697 \def\bbl@tempb##1% elt for \bbl@captionslist
2698 \ifx##1\@empty\else
2699 \bbl@exp{%
2700 \SetString\##1{%
2701 \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}%
2702 \expandafter\bbl@tempb
2703 \fi}%
2704 \expandafter\bbl@tempb\bbl@captionslist\@empty
2705 \else
2706 \ifx\bbl@initoload\relax
2707 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2708 \else
2709 \bbl@read@ini{\bbl@initoload}2% % Same
2710 \fi
2711 \fi
2712 \StartBabelCommands*{#1}{date}%

```

```

2713 \ifx\bbbl@KVP@date\@nnil
2714 \bbbl@exp{%
2715 \\\SetString\\\today{\\\bbbl@nocaption{today}{#1today}}}%
2716 \else
2717 \bbbl@savetoday
2718 \bbbl@savestate
2719 \fi
2720 \bbbl@endcommands
2721 \bbbl@load@basic{#1}%
2722 % == hyphenmins == (only if new)
2723 \bbbl@exp{%
2724 \gdef\<#1hyphenmins>{%
2725 {\bbbl@ifunset{bbbl@lfthm@#1}{2}{\bbbl@cs{lfthm@#1}}}%
2726 {\bbbl@ifunset{bbbl@rgthm@#1}{3}{\bbbl@cs{rgthm@#1}}}}}%
2727 % == hyphenrules (also in renew) ==
2728 \bbbl@provide@hyphens{#1}%
2729 \ifx\bbbl@KVP@main\@nnil\else
2730 \expandafter\main@language\expandafter{#1}%
2731 \fi}
2732 %
2733 \def\bbbl@provide@renew#1{%
2734 \ifx\bbbl@KVP@captions\@nnil\else
2735 \StartBabelCommands*{#1}{captions}%
2736 \bbbl@read@ini{\bbbl@KVP@captions}2% % Here all letters cat = 11
2737 \EndBabelCommands
2738 \fi
2739 \ifx\bbbl@KVP@date\@nnil\else
2740 \StartBabelCommands*{#1}{date}%
2741 \bbbl@savetoday
2742 \bbbl@savestate
2743 \EndBabelCommands
2744 \fi
2745 % == hyphenrules (also in new) ==
2746 \ifx\bbbl@lbfkflag\@empty
2747 \bbbl@provide@hyphens{#1}%
2748 \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.)

```

2749 \def\bbbl@load@basic#1{%
2750 \ifcase\bbbl@howloaded\or\or
2751 \ifcase\csname bbl@llevel@\language\endcsname
2752 \bbbl@csarg\let\lname@\language\relax
2753 \fi
2754 \fi
2755 \bbbl@ifunset{bbbl@lname@#1}%
2756 {\def\BabelBeforeIni##1##2{%
2757 \begingroup
2758 \let\bbbl@ini@captions@aux\@gobbletwo
2759 \def\bbbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2760 \bbbl@read@ini{##1}1%
2761 \ifx\bbbl@initoload\relax\endinput\fi
2762 \endgroup}%
2763 \begingroup % boxed, to avoid extra spaces:
2764 \ifx\bbbl@initoload\relax
2765 \bbbl@input@texini{##1}%
2766 \else
2767 \setbox\z@\hbox{\BabelBeforeIni{\bbbl@initoload}}}%
2768 \fi
2769 \endgroup}%
2770 {}}

```

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.

```

2771 \def\bbl@provide@hyphens#1{%
2772   \@tempcnta\m@ne % a flag
2773   \ifx\bbl@KVP@hyphenrules\@nnil\else
2774     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2775     \bbl@foreach\bbl@KVP@hyphenrules{%
2776       \ifnum\@tempcnta=\m@ne % if not yet found
2777         \bbl@ifsamestring{##1}{+}%
2778         {\bbl@carg\addlanguage{l@##1}}%
2779         {}%
2780         \bbl@ifunset{l@##1}% After a possible +
2781         {}%
2782         {\@tempcnta\@nameuse{l@##1}}%
2783       \fi}%
2784     \ifnum\@tempcnta=\m@ne
2785       \bbl@warning{%
2786         Requested 'hyphenrules' for '\language' not found:\%
2787         \bbl@KVP@hyphenrules.\%
2788         Using the default value. Reported}%
2789     \fi
2790   \fi
2791   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2792     \ifx\bbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2793       \bbl@ifunset{\bbl@hyphr@#1}{}% use value in ini, if exists
2794       {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2795        {}%
2796        {\bbl@ifunset{l@#1\bbl@cl{hyphr}}}%
2797        {}% if hyphenrules found:
2798        {\@tempcnta\@nameuse{l@#1\bbl@cl{hyphr}}}}%
2799     \fi
2800   \fi
2801   \bbl@ifunset{l@#1}%
2802   {\ifnum\@tempcnta=\m@ne
2803     \bbl@carg\adddialect{l@#1}\language
2804   \else
2805     \bbl@carg\adddialect{l@#1}\@tempcnta
2806   \fi}%
2807   {\ifnum\@tempcnta=\m@ne\else
2808     \global\bbl@carg\chardef{l@#1}\@tempcnta
2809   \fi}}

```

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

```

2810 \def\bbl@input@texini#1{%
2811   \bbl@bsphack
2812   \bbl@exp{%
2813     \catcode`\\=14 \catcode`\\=0
2814     \catcode`\\{=1 \catcode`\\}=2
2815     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2816     \catcode`\\=\the\catcode`\% \relax
2817     \catcode`\\=\the\catcode`\% \relax
2818     \catcode`\\{=\the\catcode`\% \relax
2819     \catcode`\\}= \the\catcode`\% \relax}%
2820   \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.

```

2821 \def\bbl@iniline#1\bbl@iniline{%
2822   \@ifnextchar[\bbl@iniset{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2823 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2824 \def\bbl@iniskip#1\@@{% if starts with ;
2825 \def\bbl@inistore#1=#2\@@{% full (default)
2826   \bbl@trim@def\bbl@tempa{#1}%

```



```

2827 \bbl@trim\toks@{#2}%
2828 \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2829 \ifin@else
2830 \bbl@xin@{,identification/include.}%
2831 {,\bbl@section/\bbl@tempa}%
2832 \ifin@\xdef\bbl@included@inis{\the\toks@}\fi
2833 \bbl@exp{%
2834 \\\g@addto@macro\\\bbl@inidata{%
2835 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2836 \fi}
2837 \def\bbl@inistore@min#1=#2\@{% minimal (maybe set in \bbl@read@ini)
2838 \bbl@trim@def\bbl@tempa{#1}%
2839 \bbl@trim\toks@{#2}%
2840 \bbl@xin@{.identification.}{.\bbl@section.}%
2841 \ifin@
2842 \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2843 \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2844 \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.

```

2845 \def\bbl@loop@ini{%
2846 \loop
2847 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2848 \endlinechar\m@ne
2849 \read\bbl@readstream to \bbl@line
2850 \endlinechar\^^M
2851 \ifx\bbl@line\@empty\else
2852 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2853 \fi
2854 \repeat}
2855 \ifx\bbl@readstream\@undefined
2856 \csname newread\endcsname\bbl@readstream
2857 \fi
2858 \def\bbl@read@ini#1#2{%
2859 \global\let\bbl@extend@ini\@gobble
2860 \openin\bbl@readstream=babel-#1.ini
2861 \ifeof\bbl@readstream
2862 \bbl@error
2863 {There is no ini file for the requested language\\%
2864 (#1: \language). Perhaps you misspelled it or your\\%
2865 installation is not complete.}%
2866 {Fix the name or reinstall babel.}%
2867 \else
2868 % == Store ini data in \bbl@inidata ==
2869 \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
2870 \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2871 \bbl@info{Importing
2872 \ifcase#2font and identification \or basic \fi
2873 data for \language\\%
2874 from babel-#1.ini. Reported}%
2875 \ifnum#2=\z@
2876 \global\let\bbl@inidata\@empty
2877 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2878 \fi
2879 \def\bbl@section{identification}%
2880 \bbl@exp{\\\bbl@inistore tag.ini=#1\\ \@}%
2881 \bbl@inistore load.level=#2\@
2882 \bbl@loop@ini

```

```

2883 % == Process stored data ==
2884 \bbl@csarg\xdef\lini@{language}{#1}%
2885 \bbl@read@ini@aux
2886 % == 'Export' data ==
2887 \bbl@ini@exports{#2}%
2888 \global\bbl@csarg\let{inidata@{language}}\bbl@inidata
2889 \global\let\bbl@inidata@empty
2890 \bbl@exp{\bbl@add@list{\bbl@ini@loaded{language}}}%
2891 \bbl@tglobal\bbl@ini@loaded
2892 \fi
2893 \closein\bbl@readstream}
2894 \def\bbl@read@ini@aux{%
2895 \let\bbl@savestrings@empty
2896 \let\bbl@savetoday@empty
2897 \let\bbl@savestate@empty
2898 \def\bbl@elt##1##2##3{%
2899 \def\bbl@section{##1}%
2900 \in{=date.}{=##1}% Find a better place
2901 \ifin@
2902 \bbl@ifunset{bbl@inikv@##1}%
2903 {\bbl@ini@calendar{##1}}%
2904 {}%
2905 \fi
2906 \bbl@ifunset{bbl@inikv@##1}{}%
2907 {\csname bbl@inikv@##1\endcsname{##2}{##3}}%
2908 \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.

```

2909 \def\bbl@extend@ini@aux#1{%
2910 \bbl@startcommands*{#1}{captions}%
2911 % Activate captions/... and modify exports
2912 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2913 \setlocalecaption{#1}{##1}{##2}}%
2914 \def\bbl@inikv@captions##1##2{%
2915 \bbl@ini@captions@aux{##1}{##2}}%
2916 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2917 \def\bbl@exportkey##1##2##3{%
2918 \bbl@ifunset{bbl@kv@##2}{}%
2919 {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2920 \bbl@exp{\global\let<bbl@##1@{language}>\<bbl@kv@##2>}}%
2921 \fi}}%
2922 % As with \bbl@read@ini, but with some changes
2923 \bbl@read@ini@aux
2924 \bbl@ini@exports\tw@
2925 % Update inidata@lang by pretending the ini is read.
2926 \def\bbl@elt##1##2##3{%
2927 \def\bbl@section{##1}%
2928 \bbl@iniline##2=##3\bbl@iniline}%
2929 \csname bbl@inidata@#1\endcsname
2930 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2931 \StartBabelCommands*{#1}{date}% And from the import stuff
2932 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2933 \bbl@savetoday
2934 \bbl@savestate
2935 \bbl@endcommands}

```

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

```

2936 \def\bbl@ini@calendar#1{%
2937 \lowercase{\def\bbl@tempa{=#1=}}%
2938 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2939 \bbl@replace\bbl@tempa{=date.}{}%
2940 \in{.licr}{#1}%
2941 \ifin@

```

```

2942 \ifcase\bbbl@engine
2943 \bbbl@replace\bbbl@tempa{.licr={}}}%
2944 \else
2945 \let\bbbl@tempa\relax
2946 \fi
2947 \fi
2948 \ifx\bbbl@tempa\relax\else
2949 \bbbl@replace\bbbl@tempa{=}{}}%
2950 \ifx\bbbl@tempa\@empty\else
2951 \xdef\bbbl@calendars{\bbbl@calendars,\bbbl@tempa}%
2952 \fi
2953 \bbbl@exp{%
2954 \def<\bbbl@inikv@#1>####1####2{%
2955 \\\bbbl@inidate###1...\relax{####2}{\bbbl@tempa}}}%
2956 \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 `\bbbl@inistore` above).

```

2957 \def\bbbl@renewinikey#1/#2\@#3{%
2958 \edef\bbbl@tempa{\zap@space #1 \@empty}% section
2959 \edef\bbbl@tempb{\zap@space #2 \@empty}% key
2960 \bbbl@trim\toks@{#3}% value
2961 \bbbl@exp{%
2962 \edef\\bbbl@key@list{\bbbl@key@list \bbbl@tempa/\bbbl@tempb;}%
2963 \\\g@addto@macro\\bbbl@inidata{%
2964 \\\bbbl@elt{\bbbl@tempa}{\bbbl@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.

```

2965 \def\bbbl@exportkey#1#2#3{%
2966 \bbbl@ifunset{\bbbl@kv@#2}%
2967 {\bbbl@csarg\gdef{#1@\language@name}{#3}}%
2968 {\expandafter\ifx\csname \bbbl@kv@#2\endcsname\@empty
2969 \bbbl@csarg\gdef{#1@\language@name}{#3}}%
2970 \else
2971 \bbbl@exp{\global\let<\bbbl@#1@\language@name><\bbbl@kv@#2>}%
2972 \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 `\bbbl@ini@exports` is called always (via `\bbbl@inisec`), while `\bbbl@after@ini` must be called explicitly after `\bbbl@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.

```

2973 \def\bbbl@iniwarning#1{%
2974 \bbbl@ifunset{\bbbl@kv@identification.warning#1}{}%
2975 {\bbbl@warning{%
2976 From babel-\bbbl@cs{lini@\language@name}.ini:\%
2977 \bbbl@cs{@kv@identification.warning#1}\%
2978 Reported }}}
2979 %
2980 \let\bbbl@release@transforms\@empty
2981 \def\bbbl@ini@exports#1{%
2982 % Identification always exported
2983 \bbbl@iniwarning}%
2984 \ifcase\bbbl@engine
2985 \bbbl@iniwarning{.pdflatex}%
2986 \or
2987 \bbbl@iniwarning{.lualatex}%
2988 \or
2989 \bbbl@iniwarning{.xelatex}%
2990 \fi%

```

```

2991 \bbl@exportkey{llevel}{identification.load.level}{}%
2992 \bbl@exportkey{elname}{identification.name.english}{}%
2993 \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2994   {csname bbl@elname@languagename\endcsname}}%
2995 \bbl@exportkey{tbcpl}{identification.tag.bcp47}{}%
2996 % Somewhat hackish. TODO
2997 \bbl@exportkey{casing}{identification.tag.bcp47}{}%
2998 \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2999 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3000 \bbl@exportkey{esname}{identification.script.name}{}%
3001 \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
3002   {csname bbl@esname@languagename\endcsname}}%
3003 \bbl@exportkey{sbcpl}{identification.script.tag.bcp47}{}%
3004 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3005 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3006 \bbl@exportkey{vbcpl}{identification.variant.tag.bcp47}{}%
3007 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
3008 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
3009 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
3010 % Also maps bcp47 -> languagename
3011 \ifbbl@bcptname
3012   \bbl@csarg\xdef{bcp@map@bbl@cl{tbcpl}}{languagename}%
3013 \fi
3014 % Conditional
3015 \ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
3016   \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3017   \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3018   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3019   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3020   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3021   \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3022   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3023   \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3024   \bbl@exportkey{intsp}{typography.intraspaces}{u}%
3025   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3026   \bbl@exportkey{chrng}{characters.ranges}{}%
3027   \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3028   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3029   \ifnum#1=\tw@ % only (re)new
3030     \bbl@exportkey{rqtex}{identification.require.babel}{}%
3031     \bbl@toglobal\bbl@savetoday
3032     \bbl@toglobal\bbl@savestate
3033     \bbl@savestrings
3034   \fi
3035 \fi}

```

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

```

3036 \def\bbl@inikv#1#2{%      key=value
3037   \toks@{#2}%             This hides #'s from ini values
3038   \bbl@csarg\edef{@kv@bbl@section.#1}{\the\toks@}}

```

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

```

3039 \let\bbl@inikv@identification\bbl@inikv
3040 \let\bbl@inikv@date\bbl@inikv
3041 \let\bbl@inikv@typography\bbl@inikv
3042 \let\bbl@inikv@characters\bbl@inikv
3043 \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 \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

3044 \def\bbl@inikv@counters#1#2{%
3045   \bbl@ifsamestring{#1}{digits}%
3046   {\bbl@error{The counter name 'digits' is reserved for mapping\\%

```

```

3047             decimal digits}%
3048         {Use another name.}}%
3049     {}%
3050 \def\bbl@tempc{#1}%
3051 \bbl@trim@def{\bbl@tempb*}{#2}%
3052 \in@{.1$}{#1$}%
3053 \ifin@
3054     \bbl@replace\bbl@tempc{.1}{}%
3055     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}\bbl@tempc}%
3056     \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3057 \fi
3058 \in@{.F.}{#1}%
3059 \ifin@else\in@{.S.}{#1}\fi
3060 \ifin@
3061     \bbl@csarg\protected@xdef{cntr@#1@\language}\bbl@tempb*}%
3062 \else
3063     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3064     \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3065     \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3066 \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:

```

3067 \ifcase\bbl@engine
3068     \bbl@csarg\def{inikv@captions.licr}#1#2{%
3069         \bbl@ini@captions@aux{#1}{#2}}
3070 \else
3071     \def\bbl@inikv@captions#1#2{%
3072         \bbl@ini@captions@aux{#1}{#2}}
3073 \fi

```

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

```

3074 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3075     \bbl@replace\bbl@tempa{.template}{}%
3076     \def\bbl@toreplace{#1}{}%
3077     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3078     \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3079     \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3080     \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}%
3081     \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3082     \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3083     \ifin@
3084         \@nameuse{\bbl@patch\bbl@tempa}%
3085         \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3086     \fi
3087     \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3088     \ifin@
3089         \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3090         \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3091             \bbl@ifunset{\bbl@bbl@tempa fmt@\language}%
3092             {[fnum@\bbl@tempa]}%
3093             {\@nameuse{\bbl@bbl@tempa fmt@\language}}}%
3094     \fi}
3095 \def\bbl@ini@captions@aux#1#2{%
3096     \bbl@trim@def\bbl@tempa{#1}%
3097     \bbl@xin@{.template}{\bbl@tempa}%
3098     \ifin@
3099         \bbl@ini@captions@template{#2}\language
3100     \else
3101         \bbl@ifblank{#2}%
3102         {\bbl@exp{%
3103             \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3104         {\bbl@trim\toks@{#2}}}%

```

```

3105 \bbl@exp{%
3106   \\bbl@add\\bbl@savestrings{%
3107     \\SetString\<\bbl@tempa name>\the\toks@}}}%
3108 \toks@expandafter{\bbl@captionslist}%
3109 \bbl@exp{\\in@{\<\bbl@tempa name>}\the\toks@}}}%
3110 \ifin@else
3111   \bbl@exp{%
3112     \\bbl@add\<bbl@extracaps@\language name>\<\bbl@tempa name>}%
3113     \\bbl@tglobal\<bbl@extracaps@\language name>}%
3114   \fi
3115 \fi}

```

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

```

3116 \def\bbl@list@the{%
3117   part,chapter,section,subsection,subsubsection,paragraph,%
3118   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3119   table,page,footnote,mpfootnote,mpfn}
3120 \def\bbl@map@cnt#1{% #1: roman, etc, // #2: enumi, etc
3121   \bbl@ifunset{\bbl@map@#1@\language name}%
3122     {\@nameuse{#1}}%
3123     {\@nameuse{\bbl@map@#1@\language name}}}%
3124 \def\bbl@inikv@labels#1#2{%
3125   \in@{.map}{#1}%
3126   \ifin@
3127     \ifx\bbl@KVP@labels\@nnil\else
3128       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3129       \ifin@
3130         \def\bbl@tempc{#1}%
3131         \bbl@replace\bbl@tempc{.map}{}%
3132         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3133         \bbl@exp{%
3134           \gdef\<bbl@map@\bbl@tempc @\language name>%
3135             {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3136         \bbl@foreach\bbl@list@the{%
3137           \bbl@ifunset{the##1}{}%
3138           {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3139             \bbl@exp{%
3140               \\bbl@sreplace\<the##1>%
3141               {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3142               \\bbl@sreplace\<the##1>%
3143               {\<\empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3144             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3145               \toks@expandafter\expandafter\expandafter{%
3146                 \csname the##1\endcsname}%
3147               \expandafter\xdef\csname the##1\endcsname{\the\toks@}}}%
3148             \fi}}}%
3149   \fi
3150 \fi
3151 %
3152 \else
3153   %
3154   % The following code is still under study. You can test it and make
3155   % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3156   % language dependent.
3157   \in@{enumerate.}{#1}%
3158   \ifin@
3159     \def\bbl@tempa{#1}%
3160     \bbl@replace\bbl@tempa{enumerate.}{}%
3161     \def\bbl@toreplace{#2}%
3162     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3163     \bbl@replace\bbl@toreplace{{}}{\csname the}%
3164     \bbl@replace\bbl@toreplace{}}{\endcsname{}}}%
3165     \toks@expandafter{\bbl@toreplace}%

```

```

3166 % TODO. Execute only once:
3167 \bbl@exp{%
3168   \\bbl@add<extras\languagename>{%
3169     \\babel@save<labelenum\romannumeral\bbl@tempa>%
3170     \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3171   \\bbl@tglobal<extras\languagename>}%
3172 \fi
3173 \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.

```

3174 \def\bbl@chapttype{chapter}
3175 \ifx\@makechapterhead\undefined
3176   \let\bbl@patchchapter\relax
3177 \else\ifx\thechapter\undefined
3178   \let\bbl@patchchapter\relax
3179 \else\ifx\ps@headings\undefined
3180   \let\bbl@patchchapter\relax
3181 \else
3182   \def\bbl@patchchapter{%
3183     \global\let\bbl@patchchapter\relax
3184     \gdef\bbl@chfmt{%
3185       \bbl@ifunset{bbl@bbl@chapttype fmt@languagename}%
3186       {\@chapapp\space\thechapter}
3187       {\@nameuse{bbl@bbl@chapttype fmt@languagename}}}
3188     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3189     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3190     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3191     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3192     \bbl@tglobal\appendix
3193     \bbl@tglobal\ps@headings
3194     \bbl@tglobal\chaptermark
3195     \bbl@tglobal\@makechapterhead}
3196   \let\bbl@patchappendix\bbl@patchchapter
3197 \fi\fi\fi
3198 \ifx\@part\undefined
3199   \let\bbl@patchpart\relax
3200 \else
3201   \def\bbl@patchpart{%
3202     \global\let\bbl@patchpart\relax
3203     \gdef\bbl@partformat{%
3204       \bbl@ifunset{bbl@partfmt@languagename}%
3205       {\partname\nobreakspace\thepart}
3206       {\@nameuse{bbl@partfmt@languagename}}}
3207     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3208     \bbl@tglobal\@part}
3209 \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

```

3210 \let\bbl@calendar\@empty
3211 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3212 \def\bbl@localedate#1#2#3#4{%
3213   \begingroup
3214     \edef\bbl@they{#2}%
3215     \edef\bbl@them{#3}%
3216     \edef\bbl@thed{#4}%
3217     \edef\bbl@tempe{%
3218       \bbl@ifunset{bbl@calpr@languagename}{\bbl@cl{calpr}},%
3219       #1}%
3220     \bbl@replace\bbl@tempe{ }{}%
3221     \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish

```

```

3222 \bbl@replace\bbl@tempe{convert}{convert=%}
3223 \let\bbl@ld@calendar\@empty
3224 \let\bbl@ld@variant\@empty
3225 \let\bbl@ld@convert\relax
3226 \def\bbl@tempb##1=##2\@{\@namedef{\bbl@ld@##1}{##2}}%
3227 \bbl@foreach\bbl@tempe{\bbl@tempb##1\@}%
3228 \bbl@replace\bbl@ld@calendar{\gregorian}{}%
3229 \ifx\bbl@ld@calendar\@empty\else
3230 \ifx\bbl@ld@convert\relax\else
3231 \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3232 {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3233 \fi
3234 \fi
3235 \@nameuse{\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3236 \edef\bbl@calendar{% Used in \month..., too
3237 \bbl@ld@calendar
3238 \ifx\bbl@ld@variant\@empty\else
3239 .\bbl@ld@variant
3240 \fi}%
3241 \bbl@cased
3242 {\@nameuse{\bbl@date@\language @\bbl@calendar}%
3243 \bbl@they\bbl@them\bbl@thed}%
3244 \endgroup}
3245 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3246 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3247 \bbl@trim@def\bbl@tempa{#1.#2}%
3248 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3249 {\bbl@trim@def\bbl@tempa{#3}%
3250 \bbl@trim\toks@{#5}%
3251 \@temptokena\expandafter{\bbl@savedate}%
3252 \bbl@exp{% Reverse order - in ini last wins
3253 \def\\bbl@savedate{%
3254 \\SetString<\month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3255 \the@temptokena}}}%
3256 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3257 {\lowercase{\def\bbl@tempb{#6}}}%
3258 \bbl@trim@def\bbl@toreplace{#5}%
3259 \bbl@TG@@date
3260 \global\bbl@csarg\let{date@\language @\bbl@tempb}\bbl@toreplace
3261 \ifx\bbl@savetoday\@empty
3262 \bbl@exp{% TODO. Move to a better place.
3263 \\AfterBabelCommands{%
3264 \def<\language date>{\\protect<\language date >}%
3265 \\newcommand<\language date >[4][{}%
3266 \\bbl@usedategroupttrue
3267 <\bbl@ensure@\language >%
3268 \\localdate[####1]{####2}{####3}{####4}}}%
3269 \def\\bbl@savetoday{%
3270 \\SetString\\today{%
3271 <\language date>[convert]%
3272 {\the\year}{\the\month}{\the\day}}}%
3273 \fi}%
3274 {}}}}

```

Dates will require some macros for the basic formatting. They may be redefined by language, so “semi-public” names (camel case) are used. Oddly enough, the CLDR places particles like “de” inconsistently in either in the date or in the month name. Note after `\bbl@replace\toks@` contains the resulting string, which is used by `\bbl@replace@finish@iii` (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3275 \let\bbl@calendar\@empty
3276 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3277 \@nameuse{\bbl@ca@#2}#1\@}%
3278 \newcommand\babelDateSpace{\nobreakspace}

```



```

3279 \newcommand\BabelDateDot{\@} % TODO. \let instead of repeating
3280 \newcommand\BabelDated[1]{\number#1}
3281 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3282 \newcommand\BabelDateM[1]{\number#1}
3283 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3284 \newcommand\BabelDateMMM[1]{\%
3285 \csname month\romannumeral#1\bbbl@calendar name\endcsname}%
3286 \newcommand\BabelDatey[1]{\number#1}%
3287 \newcommand\BabelDateyy[1]{\%
3288 \ifnum#1<10 0\number#1 %
3289 \else\ifnum#1<100 \number#1 %
3290 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3291 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3292 \else
3293 \bbbl@error
3294 {Currently two-digit years are restricted to the\
3295 range 0-9999.}%
3296 {There is little you can do. Sorry.}%
3297 \fi\fi\fi\fi}
3298 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3299 \def\bbbl@replace@finish@iii#1{%
3300 \bbbl@exp{\def\#1####1####2####3{\the\toks@}}
3301 \def\bbbl@TG@date{%
3302 \bbbl@replace\bbbl@toreplace{[ ]}{\BabelDateSpace{}}%
3303 \bbbl@replace\bbbl@toreplace{[.]}{\BabelDateDot{}}%
3304 \bbbl@replace\bbbl@toreplace{[d]}{\BabelDated{####3}}%
3305 \bbbl@replace\bbbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3306 \bbbl@replace\bbbl@toreplace{[M]}{\BabelDateM{####2}}%
3307 \bbbl@replace\bbbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3308 \bbbl@replace\bbbl@toreplace{[MMM]}{\BabelDateMMM{####2}}%
3309 \bbbl@replace\bbbl@toreplace{[y]}{\BabelDatey{####1}}%
3310 \bbbl@replace\bbbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3311 \bbbl@replace\bbbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3312 \bbbl@replace\bbbl@toreplace{[U]}{\BabelDateU{####1}}%
3313 \bbbl@replace\bbbl@toreplace{[y]}{\bbbl@datecctr[####1]}%
3314 \bbbl@replace\bbbl@toreplace{[m]}{\bbbl@datecctr[####2]}%
3315 \bbbl@replace\bbbl@toreplace{[d]}{\bbbl@datecctr[####3]}%
3316 \bbbl@replace@finish@iii\bbbl@toreplace}
3317 \def\bbbl@datecctr{\expandafter\bbbl@xdatecctr\expandafter}
3318 \def\bbbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3319 \let\bbbl@release@transforms\@empty
3320 \bbbl@csarg\let{inikv@transforms.prehyphenation}\bbbl@inikv
3321 \bbbl@csarg\let{inikv@transforms.posthyphenation}\bbbl@inikv
3322 \def\bbbl@transforms@aux#1#2#3#4,#5\relax{%
3323 #1[#2]{#3}{#4}{#5}}
3324 \begingroup % A hack. TODO. Don't require an specific order
3325 \catcode\%=12
3326 \catcode\&=14
3327 \gdef\bbbl@transforms#1#2#3{\&%
3328 \directlua{
3329 local str = [=[#2]=]
3330 str = str:gsub('%.%d+%.%d+$', '')
3331 token.set_macro('babeltempa', str)
3332 }&%
3333 \def\babeltempc{}&%
3334 \bbbl@xin@{,\babeltempa,}{,\bbbl@KVP@transforms,}&%
3335 \ifin@ \else
3336 \bbbl@xin@{: \babeltempa,}{,\bbbl@KVP@transforms,}&%
3337 \fi
3338 \ifin@
3339 \bbbl@foreach\bbbl@KVP@transforms{\&%

```

```

3340 \bbl@xin@{: \babeltempa,}{,##1,}&%
3341 \ifin@ &% font:font:transform syntax
3342 \directlua{
3343     local t = {}
3344     for m in string.gmatch('##1'..' ':'(.)') do
3345         table.insert(t, m)
3346     end
3347     table.remove(t)
3348     token.set_macro('babeltempc', ', fonts=' .. table.concat(t, ' '))
3349 }&%
3350 \fi}&%
3351 \in@{.0$}{#2$}&%
3352 \ifin@
3353 \directlua{&% (\attribute) syntax
3354     local str = string.match([[ \bbl@KVP@transforms]],
3355         '%([^(%-)]-)[^)]-\babeltempa')
3356     if str == nil then
3357         token.set_macro('babeltempb', '')
3358     else
3359         token.set_macro('babeltempb', ', attribute=' .. str)
3360     end
3361 }&%
3362 \toks@{#3}&%
3363 \bbl@exp{&%
3364     \\@g@addto@macro\\ \bbl@release@transforms{&%
3365         \relax &% Closes previous \bbl@transforms@aux
3366         \\ \bbl@transforms@aux
3367         \\ #1{label=\babeltempa\babeltempb\babeltempc}&%
3368         {\language\name}{\the\toks@}}&%
3369 \else
3370     \g@addto@macro \bbl@release@transforms{, {#3}}&%
3371 \fi
3372 \fi}
3373 \endgroup

```

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

```

3374 \def\bbl@provide@lsys#1{%
3375     \bbl@ifunset{\bbl@lname@#1}%
3376     {\bbl@load@info{#1}}%
3377     {%
3378     \bbl@csarg\let{lsys@#1}\@empty
3379     \bbl@ifunset{\bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}}%
3380     \bbl@ifunset{\bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}}%
3381     \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3382     \bbl@ifunset{\bbl@lname@#1}{%
3383         {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3384     \ifcase\bbl@engine\or\or
3385         \bbl@ifunset{\bbl@prehc@#1}{%
3386             {\bbl@exp{\\ \bbl@ifblank{\bbl@cs{prehc@#1}}}%
3387             {%
3388             {\ifx\bbl@xenoxyph\undefined
3389                 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3390                 \ifx\AtBeginDocument\notprerr
3391                     \expandafter\@secondoftwo % to execute right now
3392                 \fi
3393                 \AtBeginDocument{%
3394                     \bbl@patchfont{\bbl@xenoxyph}%
3395                     \expandafter\select@language\expandafter{\language\name}}%
3396                 \fi}}%
3397         \fi
3398         \bbl@csarg\bbl@toglobal{lsys@#1}}
3399 \def\bbl@xenoxyph@d{%

```

```

3400 \bbl@ifset{\bbl@prehc\@language\name}%
3401   {\ifnum\hyphenchar\font=\default\hyphenchar
3402     \iffontchar\font\bbl@cl{prehc}\relax
3403       \hyphenchar\font\bbl@cl{prehc}\relax
3404     \else\iffontchar\font"200B
3405       \hyphenchar\font"200B
3406     \else
3407       \bbl@warning
3408         {Neither 0 nor ZERO WIDTH SPACE are available\\%
3409           in the current font, and therefore the hyphen\\%
3410           will be printed. Try changing the fontspec's\\%
3411           'HyphenChar' to another value, but be aware\\%
3412           this setting is not safe (see the manual).\\%
3413           Reported}%
3414       \hyphenchar\font\default\hyphenchar
3415     \fi\fi
3416   \fi}%
3417   {\hyphenchar\font\default\hyphenchar}}
3418 % \fi}

```

```

3419 \def\bbl@load@info#1{%
3420   \def\BabelBeforeIni##1##2{%
3421     \begingroup
3422       \bbl@read@ini{##1}0%
3423       \endinput           % babel- .tex may contain onlypreamble's
3424     \endgroup}%          boxed, to avoid extra spaces:
3425   {\bbl@input@texini{#1}}}
```

```

3426 \def\bbl@setdigits#1#2#3#4#5{%
3427   \bbl@exp{%
3428     \def\<\language name digits>####1{%       ie, \langdigits
3429       \<\bbl@digits@\language name>####1\\\@nil}%
3430       \let\<\bbl@cnt@digits@\language name>\<\language name digits>%
3431       \def\<\language name counter>####1{%       ie, \langcounter
3432         \\\expandafter\<\bbl@counter@\language name>%
3433         \\\csname c@####1\endcsname}%
3434       \def\<\bbl@counter@\language name>####1{% ie, \bbl@counter@lang
3435         \\\expandafter\<\bbl@digits@\language name>%
3436         \\\number####1\\\@nil}}}%
3437   \def\bbl@tempa##1##2##3##4##5{%
3438     \bbl@exp{%       Wow, quite a lot of hashes! :- (
3439       \def\<\bbl@digits@\language name>#####1{%
3440         \\\ifx#####1\\\@nil                % ie, \bbl@digits@lang
3441         \\\else
3442           \\\ifx0#####1#1%
3443           \\\else\\\ifx1#####1#2%
3444           \\\else\\\ifx2#####1#3%
3445           \\\else\\\ifx3#####1#4%
3446           \\\else\\\ifx4#####1#5%
3447           \\\else\\\ifx5#####1##1%
3448           \\\else\\\ifx6#####1##2%
3449           \\\else\\\ifx7#####1##3%
3450           \\\else\\\ifx8#####1##4%
3451           \\\else\\\ifx9#####1##5%
3452           \\\else#####1%
3453           \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3454           \\\expandafter\<\bbl@digits@\language name>%

```

```

3455     \\fi}}}%
3456 \bbl@tempa}

```

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

```

3457 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3458 \ifx\\#1% % \\ before, in case #1 is multiletter
3459 \bbl@exp{%
3460 \def\\bbl@tempa###1{%
3461 \<ifcase>####1\space\the\toks@\<else>\\@ctrerr\<fi>}}%
3462 \else
3463 \toks@\expandafter{\the\toks@\or #1}%
3464 \expandafter\bbl@buildifcase
3465 \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).

```

3466 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@language}{#2}}
3467 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3468 \newcommand\localecounter[2]{%
3469 \expandafter\bbl@localecntr
3470 \expandafter{\number\csname c@#2\endcsname}{#1}}
3471 \def\bbl@alphnumeral#1#2{%
3472 \expandafter\bbl@alphnumeral@i\number#2 76543210@@{#1}}
3473 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@#9{%
3474 \ifcase\@car#8\@nil\or % Currently <10000, but prepared for bigger
3475 \bbl@alphnumeral@ii{#9}00000#1\or
3476 \bbl@alphnumeral@ii{#9}00000#1#2\or
3477 \bbl@alphnumeral@ii{#9}00000#1#2#3\or
3478 \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3479 \bbl@alphnum@invalid{>9999}%
3480 \fi}
3481 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3482 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@language}%
3483 {\bbl@cs{cntr@#1.4@language}{#5}
3484 \bbl@cs{cntr@#1.3@language}{#6}
3485 \bbl@cs{cntr@#1.2@language}{#7}
3486 \bbl@cs{cntr@#1.1@language}{#8}
3487 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3488 \bbl@ifunset{bbl@cntr@#1.S.321@language}{}%
3489 {\bbl@cs{cntr@#1.S.321@language}}%
3490 \fi}%
3491 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@language}}}
3492 \def\bbl@alphnum@invalid#1{%
3493 \bbl@error{Alphabetic numeral too large (#1)}%
3494 {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.

```

3495 \def\bbl@localeinfo#1#2{%
3496 \bbl@ifunset{bbl@info@#2}{#1}%
3497 {\bbl@ifunset{bbl@csname bbl@info@#2\endcsname @language}{#1}%
3498 {\bbl@cs{csname bbl@info@#2\endcsname @language}}}
3499 \newcommand\localeinfo[1]{%
3500 \ifx*#1\@empty % TODO. A bit hackish to make it expandable.
3501 \bbl@afterelse\bbl@localeinfo}%
3502 \else
3503 \bbl@localeinfo
3504 {\bbl@error{I've found no info for the current locale.\\%
3505 The corresponding ini file has not been loaded\\%
3506 Perhaps it doesn't exist}%
3507 {See the manual for details.}}%

```

```

3508      {#1}%
3509    \fi}
3510 % \@namedef{bbl@info@name.locale}{lcname}
3511 \@namedef{bbl@info@tag.ini}{lini}
3512 \@namedef{bbl@info@name.english}{elname}
3513 \@namedef{bbl@info@name.opentype}{lname}
3514 \@namedef{bbl@info@tag.bcp47}{tbc}
3515 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3516 \@namedef{bbl@info@tag.opentype}{lotf}
3517 \@namedef{bbl@info@script.name}{esname}
3518 \@namedef{bbl@info@script.name.opentype}{sname}
3519 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3520 \@namedef{bbl@info@script.tag.opentype}{sotf}
3521 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3522 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3523 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3524 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3525 \@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.

```

3526 \providecommand\BCPdata{}
3527 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3528   \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3529   \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3530     \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3531     {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3532     {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}%
3533   \def\bbl@bcpdata@ii#1#2{%
3534     \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3535     {\bbl@error{Unknown field '#1' in \string\BCPdata.\%
3536       Perhaps you misspelled it.}%
3537     {See the manual for details.}}%
3538     {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3539     {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3540 \fi
3541 % Still somewhat hackish. WIP.
3542 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3543 \newcommand\BabelUppercaseMapping[3]{%
3544   \let\bbl@temp\language
3545   \edef\language{#1}%
3546   \DeclareUppercaseMapping[\BCPdata{casing}]{#2}{#3}%
3547   \let\language\bbl@temp}
3548 \newcommand\BabelLowercaseMapping[3]{%
3549   \let\bbl@temp\language
3550   \edef\language{#1}%
3551   \DeclareLowercaseMapping[\BCPdata{casing}]{#2}{#3}%
3552   \let\language\bbl@temp}

```

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

```

3553 <<More package options>> \equiv
3554 \DeclareOption{ensureinfo=off}{}
3555 <</More package options>>
3556 \let\bbl@ensureinfo\@gobble
3557 \newcommand\BabelEnsureInfo{%
3558   \ifx\InputIfFileExists\@undefined\else
3559     \def\bbl@ensureinfo##1{%
3560       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3561   \fi
3562   \bbl@foreach\bbl@loaded{%
3563     \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3564     \def\language{##1}%
3565     \bbl@ensureinfo{##1}}}

```

```

3566 \@ifpackagewith{babel}{ensureinfo=off}{}%
3567 {\AtEndOfPackage{% Test for plain.
3568   \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`.

```

3569 \newcommand\getlocaleproperty{%
3570   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3571 \def\bbl@getproperty@s#1#2#3{%
3572   \let#1\relax
3573   \def\bbl@elt##1##2##3{%
3574     \bbl@ifsamestring{##1/##2}{##3}%
3575     {\providecommand#1{##3}%
3576     \def\bbl@elt####1####2####3{}}}%
3577   {}}%
3578   \bbl@cs{inidata@#2}}%
3579 \def\bbl@getproperty@x#1#2#3{%
3580   \bbl@getproperty@s{#1}{#2}{#3}%
3581   \ifx#1\relax
3582     \bbl@error
3583     {Unknown key for locale '#2':\%
3584     #3\%
3585     \string#1 will be set to \relax}%
3586     {Perhaps you misspelled it.}%
3587   \fi}
3588 \let\bbl@ini@loaded\@empty
3589 \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.

```

3590 \newcommand\babeladjust[1]{% TODO. Error handling.
3591   \bbl@forkv{#1}{%
3592     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3593     {\bbl@cs{ADJ@##1}{##2}}%
3594     {\bbl@cs{ADJ@##1@##2}}}
3595 %
3596 \def\bbl@adjust@lua#1#2{%
3597   \ifvmode
3598     \ifnum\currentgrouplevel=\z@
3599       \directlua{ Babel.#2 }%
3600       \expandafter\expandafter\expandafter\@gobble
3601     \fi
3602   \fi
3603   {\bbl@error % The error is gobbled if everything went ok.
3604     {Currently, #1 related features can be adjusted only\%
3605     in the main vertical list.}%
3606     {Maybe things change in the future, but this is what it is.}}}
3607 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3608   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3609 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3610   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3611 \@namedef{\bbl@ADJ@bidi.text@on}{%
3612   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3613 \@namedef{\bbl@ADJ@bidi.text@off}{%
3614   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3615 \@namedef{\bbl@ADJ@bidi.math@on}{%
3616   \let\bbl@noamsmath\@empty}
3617 \@namedef{\bbl@ADJ@bidi.math@off}{%
3618   \let\bbl@noamsmath\relax}
3619 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%

```

```

3620 \bbl@adjust@lua{bidi}{digits_mapped=true}}
3621 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3622 \bbl@adjust@lua{bidi}{digits_mapped=false}}
3623 %
3624 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3625 \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3626 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3627 \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3628 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3629 \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3630 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3631 \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3632 \@namedef{bbl@ADJ@justify.arabic@on}{%
3633 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3634 \@namedef{bbl@ADJ@justify.arabic@off}{%
3635 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3636 %
3637 \def\bbl@adjust@layout#1{%
3638 \ifvmode
3639 #1%
3640 \expandafter\@gobble
3641 \fi
3642 {\bbl@error % The error is gobbled if everything went ok.
3643 {Currently, layout related features can be adjusted only\\%
3644 in vertical mode.}%
3645 {Maybe things change in the future, but this is what it is.}}}
3646 \@namedef{bbl@ADJ@layout.tabular@on}{%
3647 \ifnum\bbl@tabular@mode=\tw@
3648 \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}%
3649 \else
3650 \chardef\bbl@tabular@mode\@ne
3651 \fi}
3652 \@namedef{bbl@ADJ@layout.tabular@off}{%
3653 \ifnum\bbl@tabular@mode=\tw@
3654 \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}%
3655 \else
3656 \chardef\bbl@tabular@mode\z@
3657 \fi}
3658 \@namedef{bbl@ADJ@layout.lists@on}{%
3659 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3660 \@namedef{bbl@ADJ@layout.lists@off}{%
3661 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3662 %
3663 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3664 \bbl@bcpallowedtrue}
3665 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3666 \bbl@bcpallowedfalse}
3667 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3668 \def\bbl@bcp@prefix{#1}}
3669 \def\bbl@bcp@prefix{bcp47-}
3670 \@namedef{bbl@ADJ@autoload.options}#1{%
3671 \def\bbl@autoload@options{#1}}
3672 \let\bbl@autoload@bcptoptions\@empty
3673 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3674 \def\bbl@autoload@bcptoptions{#1}}
3675 \newif\ifbbl@bcptoname
3676 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3677 \bbl@bcptonametrue}
3678 \BabelEnsureInfo}
3679 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3680 \bbl@bcptonamefalse}
3681 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3682 \directlua{ Babel.ignore_pre_char = function(node)

```

```

3683     return (node.lang == \the\csname l@nohyphenation\endcsname)
3684   end }}
3685 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3686   \directlua{ Babel.ignore_pre_char = function(node)
3687     return false
3688   end }}
3689 \@namedef{bbl@ADJ@select.write@shift}{%
3690   \let\bbl@restorelastskip\relax
3691   \def\bbl@savelastskip{%
3692     \let\bbl@restorelastskip\relax
3693     \ifvmode
3694       \ifdim\lastskip=\z@
3695         \let\bbl@restorelastskip\nobreak
3696       \else
3697         \bbl@exp{%
3698           \def\\bbl@restorelastskip{%
3699             \skip@=\the\lastskip
3700             \\nobreak \vskip-\skip@ \vskip\skip@}}%
3701       \fi
3702     \fi}}
3703 \@namedef{bbl@ADJ@select.write@keep}{%
3704   \let\bbl@restorelastskip\relax
3705   \let\bbl@savelastskip\relax}
3706 \@namedef{bbl@ADJ@select.write@omit}{%
3707   \AddBabelHook{babel-select}{beforestart}{%
3708     \expandafter\babel@aux\expandafter{\bbl@main@language}}}%
3709   \let\bbl@restorelastskip\relax
3710   \def\bbl@savelastskip##1\bbl@restorelastskip{%
3711 \@namedef{bbl@ADJ@select.encoding@off}{%
3712   \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.

```

3713 <<(*More package options)>> ≡
3714 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3715 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3716 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3717 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3718 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3719 <</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.

```

3720 \bbl@trace{Cross referencing macros}
3721 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3722   \def\@newl@bel#1#2#3{%
3723     {\@safe@activetrue
3724       \bbl@ifunset{#1#2}%
3725       \relax
3726       {\gdef\@multiplelabels{%
3727         \@latex@warning@no@line{There were multiply-defined labels}}%
3728         \@latex@warning@no@line{Label `#2' multiply defined}}%

```



```
3729 \global\@namedef{#1@#2}{#3}}
```

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

```
3730 \CheckCommand*\@testdef[3]{%
3731   \def\reserved@a{#3}%
3732   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3733   \else
3734     \@tempswatrue
3735   \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.

```
3736 \def\@testdef#1#2#3{% TODO. With @samestring?
3737   \@safe@activetrue
3738   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3739   \def\bbl@tempb{#3}%
3740   \@safe@activesfalse
3741   \ifx\bbl@tempa\relax
3742   \else
3743     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3744   \fi
3745   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3746   \ifx\bbl@tempa\bbl@tempb
3747   \else
3748     \@tempswatrue
3749   \fi}
3750 \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.

```
3751 \bbl@xin@{R}\bbl@opt@safe
3752 \ifin@
3753   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3754   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3755   {\expandafter\strip@prefix\meaning\ref}%
3756   \ifin@
3757     \bbl@redefine\@kernel@ref#1{%
3758       \@safe@activetrue\org@@kernel@ref{#1}\@safe@activesfalse}
3759     \bbl@redefine\@kernel@pageref#1{%
3760       \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3761     \bbl@redefine\@kernel@sref#1{%
3762       \@safe@activetrue\org@@kernel@sref{#1}\@safe@activesfalse}
3763     \bbl@redefine\@kernel@spageref#1{%
3764       \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3765   \else
3766     \bbl@redefinero bust\ref#1{%
3767       \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3768     \bbl@redefinero bust\pageref#1{%
3769       \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3770   \fi
3771 \else
3772   \let\org@ref\ref
3773   \let\org@pageref\pageref
3774 \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.

```
3775 \bbl@xin@{B}\bbl@opt@safe
3776 \ifin@
3777 \bbl@redefine\@citex[#1]#2{%
3778   \@safe@activestruedef\@tempa{#2}\@safe@activesfalse
3779   \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.

```
3780 \AtBeginDocument{%
3781   \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.)

```
3782   \def\@citex[#1][#2]#3{%
3783     \@safe@activestruedef\@tempa{#3}\@safe@activesfalse
3784     \org@citex[#1][#2]{\@tempa}}%
3785   }{}}
```

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

```
3786 \AtBeginDocument{%
3787   \ifpackageloaded{cite}{%
3788     \def\@citex[#1]#2{%
3789       \@safe@activestruedef\org@citex[#1][#2]\@safe@activesfalse}%
3790     }{}}
```

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

```
3791 \bbl@redefine\nocite#1{%
3792   \@safe@activestruedef\org@nocite{#1}\@safe@activesfalse}
```

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

```
3793 \bbl@redefine\bibcite{%
3794   \bbl@cite@choice
3795   \bibcite}
```

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

```
3796 \def\bbl@bibcite#1#2{%
3797   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3798 \def\bbl@cite@choice{%
3799   \global\let\bibcite\bbl@bibcite
3800   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{%
3801   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{%
3802   \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.

```
3803 \AtBeginDocument{\bbl@cite@choice}
```

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

```

3804 \bbl@redefine\@bibitem#1{%
3805   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
3806 \else
3807   \let\org@nocite\nocite
3808   \let\org@citex\citex
3809   \let\org@bibcite\bibcite
3810   \let\org@bibitem\@bibitem
3811 \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.

```

3812 \bbl@trace{Marks}
3813 \IfBabelLayout{sectioning}
3814   {\ifx\bbl@opt@headfoot\@nnil
3815     \g@addto@macro\@resetactivechars{%
3816       \set@typeset@protect
3817       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3818       \let\protect\noexpand
3819       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3820         \edef\thepage{%
3821           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3822       \fi}%
3823   \fi}
3824 {\ifbbl@single\else
3825   \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3826   \markright#1{%
3827     \bbl@ifblank{#1}%
3828     {\org@markright{}}%
3829     {\toks@{#1}%
3830      \bbl@exp{%
3831        \\org@markright{\\protect\\foreignlanguage{\language}\language}%
3832        {\\protect\\bbl@restore@actives\the\toks@}}}%

```

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

```

3833   \ifx\mkboth\markboth
3834     \def\bbl@tempc{\let\mkboth\markboth}%
3835   \else
3836     \def\bbl@tempc{%
3837       \fi
3838       \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3839       \markboth#1#2{%
3840         \protected@edef\bbl@tempb##1{%
3841           \protect\foreignlanguage
3842             {\language}\protect\bbl@restore@actives##1}%
3843         \bbl@ifblank{#1}%
3844         {\toks@{}}%
3845         {\toks@\expandafter{\bbl@tempb{#1}}}%
3846         \bbl@ifblank{#2}%
3847         {\@temptokena{}}%
3848         {\@temptokena\expandafter{\bbl@tempb{#2}}}%

```

```

3849      \bbl@exp{\org@markboth{\the\toks@}{\the\temptokena}}}%
3850      \bbl@tempc
3851    \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.

```

3852 \bbl@trace{Preventing clashes with other packages}
3853 \ifx\org@ref\undefined\else
3854   \bbl@xin@{R}\bbl@opt@safe
3855   \ifin@
3856     \AtBeginDocument{%
3857       \@ifpackageloaded{ifthen}{%
3858         \bbl@redefine@long\ifthenelse#1#2#3{%
3859           \let\bbl@temp@pref\pageref
3860           \let\pageref\org@pageref
3861           \let\bbl@temp@ref\ref
3862           \let\ref\org@ref
3863           \@safe@activestrue
3864           \org@ifthenelse{#1}%
3865             {\let\pageref\bbl@temp@pref
3866              \let\ref\bbl@temp@ref
3867              \@safe@activesfalse
3868              #2}%
3869             {\let\pageref\bbl@temp@pref
3870              \let\ref\bbl@temp@ref
3871              \@safe@activesfalse
3872              #3}%
3873           }%
3874         }{}%
3875       }
3876 \fi

```

5.3.2 varioref

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

```

3877 \AtBeginDocument{%
3878   \@ifpackageloaded{varioref}{%
3879     \bbl@redefine\@vppageref#1[#2]#3{%
3880       \@safe@activestrue
3881       \org@@@vppageref{#1}[#2][#3}%
3882       \@safe@activesfalse}%
3883     \bbl@redefine\vrefpagenum#1#2{%
3884       \@safe@activestrue

```

```

3885 \org@vrefpagenum{#1}{#2}%
3886 \@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.

```

3887 \expandafter\def\csname Ref \endcsname#1{%
3888 \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3889 }{}%
3890 }
3891 \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.

```

3892 \AtEndOfPackage{%
3893 \AtBeginDocument{%
3894 \ifpackageloaded{hhline}%
3895 {\expandafter\ifx\csname normal@char\string\endcsname\relax
3896 \else
3897 \makeatletter
3898 \def\@currname{hhline}\input{hhline.sty}\makeatother
3899 \fi}%
3900 {}}}

```

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

```

3901 \def\substitutefontfamily#1#2#3{%
3902 \lowercase{\immediate\openout15=#1#2.fd\relax}%
3903 \immediate\write15{%
3904 \string\ProvidesFile{#1#2.fd}%
3905 [\the\year/\two@digits{\the\month}/\two@digits{\the\day}}
3906 \space generated font description file]^J
3907 \string\DeclareFontFamily{#1}{#2}{ }^^J
3908 \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^^J
3909 \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^^J
3910 \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^^J
3911 \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^^J
3912 \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^^J
3913 \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^^J
3914 \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^^J
3915 \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^^J
3916 }%
3917 \closeout15
3918 }
3919 \@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

```
3920 \bbl@trace{Encoding and fonts}
3921 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3922 \newcommand\BabelNonText{TS1,T3,TS3}
3923 \let\org@TeX\TeX
3924 \let\org@LaTeX\LaTeX
3925 \let\ensureascii\@firstofone
3926 \AtBeginDocument{%
3927   \def\elt#1{,#1,}%
3928   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3929   \let\elt\relax
3930   \let\bbl@tempb\@empty
3931   \def\bbl@tempc{OT1}%
3932   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3933     \bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}}%
3934   \bbl@foreach\bbl@tempa{%
3935     \bbl@xin@{#1}{\BabelNonASCII}%
3936     \ifin@
3937       \def\bbl@tempb{#1}% Store last non-ascii
3938     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3939       \ifin@\else
3940         \def\bbl@tempc{#1}% Store last ascii
3941       \fi
3942     \fi}%
3943   \ifx\bbl@tempb\@empty\else
3944     \bbl@xin@{\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3945     \ifin@\else
3946       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3947     \fi
3948     \edef\ensureascii#1{%
3949       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3950     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3951     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3952   \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.

```
3953 \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.

```
3954 \AtBeginDocument{%
3955   \@ifpackageloaded{fontspec}%
3956   {\xdef\latinencoding{%
3957     \ifx\UTFencname\undefined
3958       EU\ifcase\bbl@engine\or2\or1\fi
3959     \else
3960       \UTFencname
3961     \fi}}%
3962   {\gdef\latinencoding{OT1}%
3963     \ifx\cf@encoding\bbl@t@one
3964       \xdef\latinencoding{\bbl@t@one}%
3965     \else
3966       \def\elt#1{,#1,}%
3967       \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3968       \let\elt\relax
3969       \bbl@xin@{,T1,}\bbl@tempa
```

```

3970      \ifin@
3971      \xdef\latinencoding{\bbl@t@one}%
3972      \fi
3973      \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.

```

3974 \DeclareRobustCommand{\latintext}{%
3975   \fontencoding{\latinencoding}\selectfont
3976   \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.

```

3977 \ifx\@undefined\DeclareTextFontCommand
3978   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3979 \else
3980   \DeclareTextFontCommand{\textlatin}{\latintext}
3981 \fi

```

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

```

3982 \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 \TeX grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua \TeX -ja` shows, vertical typesetting is possible, too.

```

3983 \bbl@trace{Loading basic (internal) bidi support}
3984 \ifodd\bbl@engine
3985 \else % TODO. Move to txtbabel
3986   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3987     \bbl@error
3988     {The bidi method 'basic' is available only in\\%
3989     luatex. I'll continue with 'bidi=default', so\\%
3990     expect wrong results}%
3991     {See the manual for further details.}%
3992     \let\bbl@beforeforeign\leavevmode
3993     \AtEndOfPackage{%
3994       \EnableBabelHook{babel-bidi}%
3995       \bbl@xebidipar}
3996   \fi\fi
3997   \def\bbl@loadxebidi#1{%
3998     \ifx\RTLfootnotetext\@undefined
3999       \AtEndOfPackage{%

```

```

4000     \EnableBabelHook{babel-bidi}%
4001     \bbl@loadfontspec % bidi needs fontspec
4002     \usepackage#1{bidi}}%
4003   \fi}
4004   \ifnum\bbl@bidimode>200 % Any xe bidi=
4005   \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4006     \bbl@tentative{bidi=bidi}
4007     \bbl@loadxebidi{}
4008   \or
4009     \bbl@loadxebidi{[rldocument]}
4010   \or
4011     \bbl@loadxebidi{}
4012   \fi
4013 \fi
4014 \fi
4015 % TODO? Separate:
4016 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
4017   \let\bbl@beforeforeign\leavevmode
4018   \ifodd\bbl@engine
4019     \newattribute\bbl@attr@dir
4020     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4021     \bbl@exp{\output{\bodydir\pagedir\the\output}}
4022   \fi
4023   \AtEndOfPackage{%
4024     \EnableBabelHook{babel-bidi}%
4025     \ifodd\bbl@engine\else
4026       \bbl@xebidipar
4027     \fi}
4028 \fi

```

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

```

4029 \bbl@trace{Macros to switch the text direction}
4030 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4031 \def\bbl@rscripts{% TODO. Base on codes ??
4032   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4033   Old Hungarian,Lydian,Mandaean,Manichaeen,%
4034   Meroitic Cursive,Meroitic,Old North Arabian,%
4035   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4036   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4037   Old South Arabian,}%
4038 \def\bbl@provide@dirs#1{%
4039   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4040   \ifin@
4041     \global\bbl@csarg\chardef{wdir@#1}\@ne
4042     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4043     \ifin@
4044       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
4045     \fi
4046   \else
4047     \global\bbl@csarg\chardef{wdir@#1}\z@
4048   \fi
4049   \ifodd\bbl@engine
4050     \bbl@csarg\ifcase{wdir@#1}%
4051       \directlua{ Babel.locale_props[\the\localeid].texdir = 'l' }%
4052     \or
4053       \directlua{ Babel.locale_props[\the\localeid].texdir = 'r' }%
4054     \or
4055       \directlua{ Babel.locale_props[\the\localeid].texdir = 'al' }%
4056     \fi
4057   \fi}
4058 \def\bbl@switchdir{%
4059   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%

```



```

4060 \bbl@ifunset{\bbl@wdir@\language}\bbl@provide@dirs{\language}}{}%
4061 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}
4062 \def\bbl@setdirs#1{% TODO - math
4063 \ifcase\bbl@select@type % TODO - strictly, not the right test
4064 \bbl@bodydir{#1}%
4065 \bbl@pdir{#1}% <- Must precede \bbl@textdir
4066 \fi
4067 \bbl@textdir{#1}}
4068 % TODO. Only if \bbl@bidimode > 0?:
4069 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4070 \DisableBabelHook{babel-bidi}

```

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

```

4071 \ifodd\bbl@engine % luatex=1
4072 \else % pdftex=0, xetex=2
4073 \newcount\bbl@dirlevel
4074 \chardef\bbl@thetextdir\z@
4075 \chardef\bbl@thepardir\z@
4076 \def\bbl@textdir#1{%
4077 \ifcase#1\relax
4078 \chardef\bbl@thetextdir\z@
4079 \bbl@textdir@i\beginL\endL
4080 \else
4081 \chardef\bbl@thetextdir@ne
4082 \bbl@textdir@i\beginR\endR
4083 \fi}
4084 \def\bbl@textdir@i#1#2{%
4085 \ifhmode
4086 \ifnum\currentgrouplevel>\z@
4087 \ifnum\currentgrouplevel=\bbl@dirlevel
4088 \bbl@error{Multiple bidi settings inside a group}%
4089 {I'll insert a new group, but expect wrong results.}%
4090 \bgroup\aftergroup#2\aftergroup\egroup
4091 \else
4092 \ifcase\currentgrouptype\or % 0 bottom
4093 \aftergroup#2% 1 simple {}
4094 \or
4095 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4096 \or
4097 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4098 \or\or\or % vbox vtop align
4099 \or
4100 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4101 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4102 \or
4103 \aftergroup#2% 14 \begingroup
4104 \else
4105 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4106 \fi
4107 \fi
4108 \bbl@dirlevel\currentgrouplevel
4109 \fi
4110 #1%
4111 \fi}
4112 \def\bbl@pdir#1{\chardef\bbl@thepardir#1\relax}
4113 \let\bbl@bodydir\@gobble
4114 \let\bbl@pagedir\@gobble
4115 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

4116 \def\bbl@xebidipar{%
4117 \let\bbl@xebidipar\relax

```

```

4118 \TeXeTstate\@ne
4119 \def\bbl@xeverypar{%
4120   \ifcase\bbl@thepardir
4121     \ifcase\bbl@thetextdir\else\beginR\fi
4122   \else
4123     {\setbox\z@\lastbox\beginR\box\z@}%
4124   \fi}%
4125 \let\bbl@severypar\everypar
4126 \newtoks\everypar
4127 \everypar=\bbl@severypar
4128 \bbl@severypar{\bbl@xeverypar\the\everypar}}
4129 \ifnum\bbl@bidimode>200 % Any xe bidi=
4130 \let\bbl@textdir@i\@gobbletwo
4131 \let\bbl@xebidipar\@empty
4132 \AddBabelHook{bidi}{foreign}{%
4133   \def\bbl@tempa{\def\BabelText###1}%
4134   \ifcase\bbl@thetextdir
4135     \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4136   \else
4137     \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4138   \fi}
4139 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4140 \fi
4141 \fi

```

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

```

4142 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4143 \AtBeginDocument{%
4144   \ifx\pdfstringdefDisableCommands\@undefined\else
4145     \ifx\pdfstringdefDisableCommands\relax\else
4146       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4147     \fi
4148   \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`.

```

4149 \bbl@trace{Local Language Configuration}
4150 \ifx\loadlocalcfg\@undefined
4151   \ifpackagewith{babel}{noconfigs}%
4152     {\let\loadlocalcfg\@gobble}%
4153   {\def\loadlocalcfg#1{%
4154     \InputIfFileExists{#1.cfg}%
4155     {\typeout{*****^J%
4156               * Local config file #1.cfg used^^J%
4157             *}}%
4158     \@empty}}
4159 \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).

```

4160 \bbl@trace{Language options}
4161 \let\bbl@afterlang\relax
4162 \let\BabelModifiers\relax
4163 \let\bbl@loaded\@empty

```

```

4164 \def\bbl@load@language#1{%
4165   \InputIfFileExists{#1.ldf}%
4166   {\edef\bbl@loaded{\CurrentOption
4167     \ifx\bbl@loaded\empty\else,\bbl@loaded\fi}%
4168     \expandafter\let\expandafter\bbl@afterlang
4169     \csname\CurrentOption.ldf-h@k\endcsname
4170     \expandafter\let\expandafter\BabelModifiers
4171     \csname bbl@mod@\CurrentOption\endcsname
4172     \bbl@exp{\AtBeginDocument}%
4173     \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4174   {\bbl@error{%
4175     Unknown option '\CurrentOption'. Either you misspelled it\%
4176     or the language definition file \CurrentOption.ldf was not found}%
4177     Valid options are, among others: shorthands=, KeepShorthandsActive,\%
4178     activeacute, activegrave, noconfigs, safe=, main=, math=\%
4179     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.

```

4180 \def\bbl@try@load@lang#1#2#3{%
4181   \IfFileExists{\CurrentOption.ldf}%
4182   {\bbl@load@language{\CurrentOption}}%
4183   {\#1\bbl@load@language{#2}#3}}
4184 %
4185 \DeclareOption{hebrew}{%
4186   \input{rlbabel.def}%
4187   \bbl@load@language{hebrew}}
4188 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4189 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4190 \DeclareOption{northersami}{\bbl@try@load@lang{}{samin}{}}
4191 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4192 \DeclareOption{polutonikogreek}{%
4193   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4194 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4195 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4196 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4197 \DeclareOption{upporsorbian}{\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.

```

4198 \ifx\bbl@opt@config\@nnil
4199   \ifpackagewith{babel}{noconfigs}{}%
4200     {\InputIfFileExists{bblopts.cfg}%
4201       {\typeout{*****^J%
4202         * Local config file bblopts.cfg used^^J%
4203         *}}}%
4204     }%
4205 \else
4206   \InputIfFileExists{\bbl@opt@config.cfg}%
4207   {\typeout{*****^J%
4208     * Local config file \bbl@opt@config.cfg used^^J%
4209     *}}}%
4210   {\bbl@error{%
4211     Local config file '\bbl@opt@config.cfg' not found}%
4212     Perhaps you misspelled it.}}%
4213 \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.

```

4214 \ifx\bbl@opt@main\@nnil
4215 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4216 \let\bbl@tempb\@empty
4217 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4218 \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4219 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4220 \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4221 \ifodd\bbl@iniflag % = *=
4222 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4223 \else % n +=
4224 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4225 \fi
4226 \fi}%
4227 \fi
4228 \else
4229 \bbl@info{Main language set with 'main='. Except if you have\\%
4230 problems, prefer the default mechanism for setting\\%
4231 the main language, ie, as the last declared.\\%
4232 Reported}
4233 \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).

```

4234 \ifx\bbl@opt@main\@nnil\else
4235 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4236 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4237 \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.

```

4238 \bbl@foreach\bbl@language@opts{%
4239 \def\bbl@tempa{#1}%
4240 \ifx\bbl@tempa\bbl@opt@main\else
4241 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4242 \bbl@ifunset{ds@#1}%
4243 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4244 {}%
4245 \else % + * (other = ini)
4246 \DeclareOption{#1}{%
4247 \bbl@ldfinit
4248 \babelprovide[import]{#1}%
4249 \bbl@afterldf{}}%
4250 \fi
4251 \fi}
4252 \bbl@foreach\@classoptionslist{%
4253 \def\bbl@tempa{#1}%
4254 \ifx\bbl@tempa\bbl@opt@main\else
4255 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4256 \bbl@ifunset{ds@#1}%
4257 {\IfFileExists{#1.ldf}%
4258 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4259 {}}%
4260 {}%
4261 \else % + * (other = ini)
4262 \IfFileExists{babel-#1.tex}%
4263 {\DeclareOption{#1}{%
4264 \bbl@ldfinit
4265 \babelprovide[import]{#1}%
4266 \bbl@afterldf{}}}%
4267 {}%

```

```

4268     \fi
4269 \fi}

```

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

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

```

4270 \def\AfterBabelLanguage#1{%
4271   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4272 \DeclareOption*{}
4273 \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.

```

4274 \bbl@trace{Option 'main'}
4275 \ifx\bbl@opt@main\@nnil
4276   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4277   \let\bbl@tempc\@empty
4278   \edef\bbl@templ{\,\bbl@loaded,}
4279   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4280   \bbl@for\bbl@tempb\bbl@tempa{%
4281     \edef\bbl@tempd{\,\bbl@tempb,}%
4282     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4283     \bbl@xin{\bbl@tempd}{\bbl@templ}%
4284     \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4285   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4286   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4287   \ifx\bbl@tempb\bbl@tempc\else
4288     \bbl@warning{%
4289       Last declared language option is '\bbl@tempc',\,%
4290       but the last processed one was '\bbl@tempb'.\,%
4291       The main language can't be set as both a global\,%
4292       and a package option. Use 'main=\bbl@tempc' as\,%
4293       option. Reported}
4294   \fi
4295 \else
4296   \ifodd\bbl@iniflag % case 1,3 (main is ini)
4297     \bbl@ldfinit
4298     \let\CurrentOption\bbl@opt@main
4299     \bbl@exp{% \bbl@opt@provide = empty if *
4300       \\ \babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4301     \bbl@afterldf{}
4302     \DeclareOption{\bbl@opt@main}{}
4303   \else % case 0,2 (main is ldf)
4304     \ifx\bbl@loadmain\relax
4305       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4306     \else
4307       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4308     \fi
4309     \ExecuteOptions{\bbl@opt@main}
4310     \@namedef{ds@\bbl@opt@main}{}%
4311   \fi
4312   \DeclareOption*{}
4313   \ProcessOptions*
4314 \fi
4315 \bbl@exp{%
4316   \\ \AtBeginDocument{\\ \bbl@usehooks@lang{/}{\begindocument}{}}}%
4317 \def\AfterBabelLanguage{%
4318   \bbl@error
4319   {Too late for \string\AfterBabelLanguage}%

```

```
4320 {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.

```
4321 \ifx\bbl@main@language\undefined
4322 \bbl@info{%
4323   You haven't specified a language as a class or package\\%
4324   option. I'll load 'nil'. Reported}
4325 \bbl@load@language{nil}
4326 \fi
4327 \</package>
```

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

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

Because plain $\text{T}_{\text{E}}\text{X}$ users might want to use some of the features of the babel system too, care has to be taken that plain $\text{T}_{\text{E}}\text{X}$ can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain $\text{T}_{\text{E}}\text{X}$ and $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, some of it is for the $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 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`

```
4328 \*kernel>
4329 \let\bbl@onlyswitch\@empty
4330 \input babel.def
4331 \let\bbl@onlyswitch\@undefined
4332 \</kernel>
4333 \*patterns>
```

7 Loading hyphenation patterns

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

```
4334 \<<Make sure ProvidesFile is defined>>
4335 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4336 \xdef\bbl@format{\jobname}
4337 \def\bbl@version{<<version>>}
4338 \def\bbl@date{<<date>>}
4339 \ifx\AtBeginDocument\undefined
4340 \def\@empty{}
4341 \fi
4342 \<<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.

```
4343 \def\process@line#1#2 #3 #4 {%
4344   \ifx=#1%
4345     \process@synonym{#2}%
4346   \else
4347     \process@language{#1#2}{#3}{#4}%
4348   \fi
4349   \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.

```

4350 \toks@{}
4351 \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.

```

4352 \def\process@synonym#1{%
4353   \ifnum\last@language=\m@ne
4354     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4355   \else
4356     \expandafter\chardef\csname l@#1\endcsname\last@language
4357     \wlog{\string\l@#1=\string\language\the\last@language}%
4358     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4359       \csname\language\hyphenmins\endcsname
4360     \let\bbl@elt\relax
4361     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4362   \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 \<lang>hyphenmins macro. When no assignments were made we provide a default setting.

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

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

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

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

\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}. Note the last 2

arguments are empty in ‘dialects’ defined in language.dat with =. Note also the language name can have encoding info.

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

```

4363 \def\process@language#1#2#3{%
4364   \expandafter\addlanguage\csname l@#1\endcsname
4365   \expandafter\language\csname l@#1\endcsname
4366   \edef\language{#1}%
4367   \bbl@hook@everylanguage{#1}%
4368   % > luatex
4369   \bbl@get@enc#1: :@@@
4370   \begingroup
4371     \lefthyphenmin\m@ne
4372     \bbl@hook@loadpatterns{#2}%
4373     % > luatex
4374     \ifnum\lefthyphenmin=\m@ne
4375     \else
4376       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4377         \the\lefthyphenmin\the\righthyphenmin}%
4378     \fi

```

```

4379 \endgroup
4380 \def\bbl@tempa{#3}%
4381 \ifx\bbl@tempa\empty\else
4382   \bbl@hook@loadexceptions{#3}%
4383   % > luatex
4384 \fi
4385 \let\bbl@elt\relax
4386 \edef\bbl@languages{%
4387   \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4388 \ifnum\the\language=\z@
4389   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4390     \set@hyphenmins\tw@\thr@@\relax
4391   \else
4392     \expandafter\expandafter\expandafter\set@hyphenmins
4393     \csname #1hyphenmins\endcsname
4394   \fi
4395   \the\toks@
4396   \toks@{}%
4397 \fi}

```

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

```

4398 \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.

```

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

```



```

4435 \let\locale\setlocale
4436 \let\selectlocale\setlocale
4437 \let\localename\setlocale
4438 \let\textlocale\setlocale
4439 \let\textlanguage\setlocale
4440 \let\languagegettext\setlocale}
4441 \begingroup
4442 \def\AddBabelHook#1#2{%
4443   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4444     \def\next{\toks1}%
4445   \else
4446     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4447   \fi
4448   \next}
4449 \ifx\directlua@undefined
4450   \ifx\XeTeXinputencoding@undefined\else
4451     \input xebabel.def
4452   \fi
4453 \else
4454   \input luababel.def
4455 \fi
4456 \openin1 = babel-\bbl@format.cfg
4457 \ifeof1
4458 \else
4459   \input babel-\bbl@format.cfg\relax
4460 \fi
4461 \closein1
4462 \endgroup
4463 \bbl@hook@loadkernel{switch.def}

```

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

```

4464 \openin1 = language.dat

```

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

```

4465 \def\language{english}%
4466 \ifeof1
4467   \message{I couldn't find the file language.dat,\space
4468     I will try the file hyphen.tex}
4469   \input hyphen.tex\relax
4470   \chardef\l@english\z@
4471 \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`.

```

4472 \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.

```

4473 \loop
4474   \endlinechar\m@ne
4475   \read1 to \bbl@line
4476   \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.

```

4477   \if T\ifeof1F\fi T\relax
4478   \ifx\bbl@line@empty\else
4479     \edef\bbl@line{\bbl@line\space\space\space}%
4480     \expandafter\process@line\bbl@line\relax

```

```

4481     \fi
4482 \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.

```

4483 \begingroup
4484   \def\bbl@elt#1#2#3#4{%
4485     \global\language=#2\relax
4486     \gdef\language#1{%
4487       \def\bbl@elt##1##2##3##4{}}%
4488   \bbl@languages
4489 \endgroup
4490 \fi
4491 \closeinl

```

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

```

4492 \if/\the\toks@/\else
4493   \errhelp{language.dat loads no language, only synonyms}
4494   \errmessage{Orphan language synonym}
4495 \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.

```

4496 \let\bbl@line\undefined
4497 \let\process@line\undefined
4498 \let\process@synonym\undefined
4499 \let\process@language\undefined
4500 \let\bbl@get@enc\undefined
4501 \let\bbl@hyph@enc\undefined
4502 \let\bbl@tempa\undefined
4503 \let\bbl@hook@loadkernel\undefined
4504 \let\bbl@hook@everylanguage\undefined
4505 \let\bbl@hook@loadpatterns\undefined
4506 \let\bbl@hook@loadexceptions\undefined
4507 \patterns

```

Here the code for `iniTeX` ends.

8 Font handling with fontspec

Add the `bid` 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 `bid` [misplaced].

```

4508 <<More package options>> ≡
4509 \chardef\bbl@bidimode\z@
4510 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4511 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4512 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4513 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4514 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4515 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4516 <</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.

```

4517 <<Font selection>> ≡
4518 \bbl@trace{Font handling with fontspec}
4519 \ifx\ExplSyntaxOn\undefined\else
4520   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro

```

```

4521 \in@{, #1,}{, no-script, language-not-exist,}%
4522 \ifin@ \else \bbl@tempfs@nx{#1}{#2}\fi}
4523 \def\bbl@fs@warn@nx#1#2#3{%
4524 \in@{, #1,}{, no-script, language-not-exist,}%
4525 \ifin@ \else \bbl@tempfs@nx{#1}{#2}{#3}\fi}
4526 \def\bbl@loadfontspec{%
4527 \let\bbl@loadfontspec\relax
4528 \ifx\fontspec\undefined
4529 \usepackage{fontspec}%
4530 \fi}%
4531 \fi
4532 \@onlypreamble\babelfont
4533 \newcommand\babelfont[2][{}% 1=langs/scripts 2=fam
4534 \bbl@foreach{#1}{%
4535 \expandafter\ifx\csname date##1\endcsname\relax
4536 \IfFileExists{babel-##1.tex}%
4537 {\babelprovide{##1}}%
4538 }%
4539 \fi}%
4540 \edef\bbl@tempa{#1}%
4541 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4542 \bbl@loadfontspec
4543 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4544 \bbl@bblfont}
4545 \newcommand\bbl@bblfont[2][{}% 1=features 2=fontname, @font=rm|sf|tt
4546 \bbl@ifunset{\bbl@tempb family}%
4547 {\bbl@providefam{\bbl@tempb}}%
4548 }%
4549 % For the default font, just in case:
4550 \bbl@ifunset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4551 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4552 {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save \bbl@rmdflt@
4553 \bbl@exp{%
4554 \let<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4555 \\\bbl@font@set<\bbl@\bbl@tempb dflt@\languagename>%
4556 <\bbl@tempb default>\<\bbl@tempb family>}}%
4557 {\bbl@foreach\bbl@tempa{% ie \bbl@rmdflt@lang / *scrt
4558 \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:

```

4559 \def\bbl@providefam#1{%
4560 \bbl@exp{%
4561 \\\newcommand<#1default>{}% Just define it
4562 \\\bbl@add@list\\bbl@font@fams{#1}%
4563 \\\DeclareRobustCommand<#1family>{%
4564 \\\not@math@alphabet<#1family>\relax
4565 % \\\prepare@family@series@update{#1}<#1default>% TODO. Fails
4566 \\\fontfamily<#1default>%
4567 <\ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4568 \\\selectfont}%
4569 \\\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.

```

4570 \def\bbl@nostdfont#1{%
4571 \bbl@ifunset{\bbl@WFF@f@family}%
4572 {\bbl@csarg\gdef{WFF@f@family}}{}% Flag, to avoid dupl warns
4573 \bbl@infowarn{The current font is not a babel standard family:\\%
4574 #1%
4575 \fontname\font\\%
4576 There is nothing intrinsically wrong with this warning, and\\%
4577 you can ignore it altogether if you do not need these\\%
4578 families. But if they are used in the document, you should be\\%
4579 aware 'babel' will not set Script and Language for them, so\\%

```

```

4580     you may consider defining a new family with \string\babelfont.\\%
4581     See the manual for further details about \string\babelfont.\\%
4582     Reported}}
4583   }}%
4584 \gdef\bbl@switchfont{%
4585   \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
4586   \bbl@exp{% eg Arabic -> arabic
4587     \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4588   \bbl@foreach\bbl@font@fams{%
4589     \bbl@ifunset{\bbl@##ldflt@\language}%      (1) language?
4590     {\bbl@ifunset{\bbl@##ldflt@*\bbl@tempa}%    (2) from script?
4591       {\bbl@ifunset{\bbl@##ldflt@}%             2=F - (3) from generic?
4592         {}%                                     123=F - nothing!
4593         {\bbl@exp{%                             3=T - from generic
4594           \global\let<\bbl@##ldflt@\language>%
4595             \<\bbl@##ldflt@>}}}%
4596         {\bbl@exp{%                             2=T - from script
4597           \global\let<\bbl@##ldflt@\language>%
4598             \<\bbl@##ldflt@*\bbl@tempa>}}}%
4599       {}%                                     1=T - language, already defined
4600   \def\bbl@tempa{\bbl@nostdfont{}}% TODO. Don't use \bbl@tempa
4601   \bbl@foreach\bbl@font@fams{% don't gather with prev for
4602     \bbl@ifunset{\bbl@##ldflt@\language}%
4603     {\bbl@cs{famrst@##1}%
4604       \global\bbl@csarg\let{famrst@##1}\relax}%
4605     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4606       \\bbl@add\\originalTeX{%
4607         \\bbl@font@rst{\bbl@cl{##ldflt}}}%
4608         \<##ldflt>\<##lfamily>{##1}}}%
4609       \\bbl@font@set<\bbl@##ldflt@\language>% the main part!
4610       \<##ldflt>\<##lfamily>}}}%
4611   \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.

```

4612 \ifx\f@family\undefined\else % if latex
4613   \ifcase\bbl@engine % if pdftex
4614     \let\bbl@ckeckstdfonts\relax
4615   \else
4616     \def\bbl@ckeckstdfonts{%
4617       \begingroup
4618       \global\let\bbl@ckeckstdfonts\relax
4619       \let\bbl@tempa\empty
4620       \bbl@foreach\bbl@font@fams{%
4621         \bbl@ifunset{\bbl@##ldflt@}%
4622         {\@nameuse{##lfamily}%
4623           \bbl@csarg\gdef{WFF@f@family}}}% Flag
4624         \bbl@exp{\\bbl@add\\bbl@tempa{* \<##lfamily>= \f@family\\}%
4625           \space\space\fontname\font\\}%
4626         \bbl@csarg\xdef{##ldflt@}{\f@family}%
4627         \expandafter\xdef\csname ##ldflt\endcsname{\f@family}%
4628         {}}%
4629       \ifx\bbl@tempa\empty\else
4630         \bbl@infowarn{The following font families will use the default\\%
4631           settings for all or some languages:\\%
4632           \bbl@tempa
4633           There is nothing intrinsically wrong with it, but\\%
4634           'babel' will no set Script and Language, which could\\%
4635           be relevant in some languages. If your document uses\\%
4636           these families, consider redefining them with \string\babelfont.\\%
4637           Reported}%
4638       \fi
4639     \endgroup}

```

```
4640 \fi
4641 \fi
```

Now the macros defining the font with fontspec.

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

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

```
4642 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4643 \bbl@xin@{<>}{#1}%
4644 \ifin@
4645 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4646 \fi
4647 \bbl@exp{%
4648 \def\\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4649 \\bbl@ifsamestring{#2}{\f@family}%
4650 {\\#3%
4651 \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}}}%
4652 \let\\bbl@tempa\relax}%
4653 {}}}}
4654 % TODO - next should be global?, but even local does its job. I'm
4655 % still not sure -- must investigate:
4656 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4657 \let\bbl@tempe\bbl@mapselect
4658 \edef\bbl@tempb{\bbl@stripslash#4/}% Catcodes hack (better pass it).
4659 \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}}}%
4660 \let\bbl@mapselect\relax
4661 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4662 \let#4\@empty % Make sure \renewfontfamily is valid
4663 \bbl@exp{%
4664 \let\\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
4665 <\keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@ccl{sname}}}%
4666 {\\newfontscript{\bbl@ccl{sname}}{\bbl@ccl{sotf}}}%
4667 <\keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@ccl{lname}}}%
4668 {\\newfontlanguage{\bbl@ccl{lname}}{\bbl@ccl{lotf}}}%
4669 \let\\bbl@tempfs@nx<__fontspec_warning:nx>%
4670 \let<__fontspec_warning:nx>\\bbl@fs@warn@nx
4671 \let\\bbl@tempfs@nxx<__fontspec_warning:nxx>%
4672 \let<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4673 \\renewfontfamily\\#4%
4674 [\bbl@ccl{lsys},%
4675 \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4676 #2]}{#3}% ie \bbl@exp{..}{#3}
4677 \bbl@exp{%
4678 \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4679 \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4680 \beginngroup
4681 #4%
4682 \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4683 \endgroup % TODO. Find better tests:
4684 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4685 {\expandafter\meaning\csize TU/#1/bx/sc\endcsize}%
4686 \ifin@
4687 \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4688 \fi
4689 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4690 {\expandafter\meaning\csize TU/#1/bx/scit\endcsize}%
4691 \ifin@
```

```

4692 \global\bbbl@carg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4693 \fi
4694 \let#4\bbbl@temp@fam
4695 \bbbl@exp{\let<\bbbl@stripslash#4\space>\bbbl@temp@pfam
4696 \let\bbbl@mapselect\bbbl@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.

```

4697 \def\bbbl@font@rst#1#2#3#4{%
4698 \bbbl@carg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}

```

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

```

4699 \def\bbbl@font@fams{rm,sf,tt}
4700 <</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.

```

4701 <<{*Footnote changes}>> ≡
4702 \bbbl@trace{Bidi footnotes}
4703 \ifnum\bbbl@bidimode>\z@ % Any bidi=
4704 \def\bbbl@footnote#1#2#3{%
4705 \ifnextchar[%
4706 {\bbbl@footnote@o{#1}{#2}{#3}}%
4707 {\bbbl@footnote@x{#1}{#2}{#3}}}
4708 \long\def\bbbl@footnote@x#1#2#3#4{%
4709 \bgroup
4710 \select@language@x{\bbbl@main@language}%
4711 \bbbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4712 \egroup}
4713 \long\def\bbbl@footnote@o#1#2#3[#4]#5{%
4714 \bgroup
4715 \select@language@x{\bbbl@main@language}%
4716 \bbbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4717 \egroup}
4718 \def\bbbl@footnotetext#1#2#3{%
4719 \ifnextchar[%
4720 {\bbbl@footnotetext@o{#1}{#2}{#3}}%
4721 {\bbbl@footnotetext@x{#1}{#2}{#3}}}
4722 \long\def\bbbl@footnotetext@x#1#2#3#4{%
4723 \bgroup
4724 \select@language@x{\bbbl@main@language}%
4725 \bbbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4726 \egroup}
4727 \long\def\bbbl@footnotetext@o#1#2#3[#4]#5{%
4728 \bgroup
4729 \select@language@x{\bbbl@main@language}%
4730 \bbbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4731 \egroup}
4732 \def\BabelFootnote#1#2#3#4{%
4733 \ifx\bbbl@fn@footnote\undefined
4734 \let\bbbl@fn@footnote\footnote
4735 \fi
4736 \ifx\bbbl@fn@footnotetext\undefined
4737 \let\bbbl@fn@footnotetext\footnotetext
4738 \fi
4739 \bbbl@ifblank{#2}%
4740 {\def#1{\bbbl@footnote{\@firstofone}{#3}{#4}}
4741 \@namedef{\bbbl@stripslash#1text}%

```

```

4742         {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4743     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}}%
4744     \@namedef{\bbl@stripslash#1text}%
4745         {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4746 \fi
4747 <</Footnote changes>>

Now, the code.

4748 (*xetex)
4749 \def\BabelStringsDefault{unicode}
4750 \let\xebbl@stop\relax
4751 \AddBabelHook{xetex}{encodedcommands}{%
4752     \def\bbl@tempa{#1}%
4753     \ifx\bbl@tempa@empty
4754         \XeTeXinputencoding"bytes"%
4755     \else
4756         \XeTeXinputencoding"#1"%
4757     \fi
4758     \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4759 \AddBabelHook{xetex}{stopcommands}{%
4760     \xebbl@stop
4761     \let\xebbl@stop\relax}
4762 \def\bbl@intraspace#1 #2 #3@@{%
4763     \bbl@csarg\gdef{xeisp@\languagename}%
4764         {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4765 \def\bbl@intrapenalty#1@@{%
4766     \bbl@csarg\gdef{xeipn@\languagename}%
4767         {\XeTeXlinebreakpenalty #1\relax}}
4768 \def\bbl@provide@intraspace{%
4769     \bbl@xin@{/s}{\bbl@cl{lbrk}}}%
4770     \ifin@else\bbl@xin@{/c}{\bbl@cl{lbrk}}}\fi
4771     \ifin@
4772         \bbl@ifunset{\bbl@intsp@\languagename}{}%
4773         {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname@empty\else
4774             \ifx\bbl@KVP@intraspace@nnil
4775                 \bbl@exp{%
4776                     \\bbl@intraspace\bbl@cl{intsp}\\\@}%
4777                 \fi
4778                 \ifx\bbl@KVP@intrapenalty@nnil
4779                     \bbl@intrapenalty0@@
4780                 \fi
4781                 \fi
4782                 \ifx\bbl@KVP@intraspace@nnil\else % We may override the ini
4783                     \expandafter\bbl@intraspace\bbl@KVP@intraspace@@
4784                 \fi
4785                 \ifx\bbl@KVP@intrapenalty@nnil\else
4786                     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
4787                 \fi
4788                 \bbl@exp{%
4789                     % TODO. Execute only once (but redundant):
4790                     \\bbl@add<extras\languagename>%
4791                     \XeTeXlinebreaklocale "\bbl@cl{tbcpr}"%
4792                     \<bbl@xeisp@\languagename>%
4793                     \<bbl@xeipn@\languagename>%
4794                     \\bbl@toglobal<extras\languagename>%
4795                     \\bbl@add<noextras\languagename>%
4796                     \XeTeXlinebreaklocale ""}%
4797                     \\bbl@toglobal<noextras\languagename>%
4798                 \ifx\bbl@ispace@size@undefined
4799                     \gdef\bbl@ispace@size{\bbl@cl{xeisp}}}%
4800                 \ifx\AtBeginDocument@notprerr
4801                     \expandafter\@secondoftwo % to execute right now
4802                 \fi

```

```

4803      \AtBeginDocument{\bbl@patchfont{\bbl@ispacefont}}%
4804      \fi}%
4805      \fi}
4806      \ifx\DisableBabelHook\undefined\endinput\fi
4807      \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4808      \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4809      \DisableBabelHook{babel-fontspec}
4810      <<Font selection>>
4811      \def\bbl@provide@extra#1{}
4812      </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: \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.

```

4813 <*xetex | texxet>
4814 \providecommand\bbl@provide@intraspace{}
4815 \bbl@trace{Redefinitions for bidi layout}
4816 \def\bbl@sspre@caption{%
4817   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4818 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4819 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4820 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4821 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4822   \def\@hangfrom#1{%
4823     \setbox\@tempboxa\hbox{#1}%
4824     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4825     \noindent\box\@tempboxa}
4826   \def\raggedright{%
4827     \let\@centercr
4828     \bbl@startskip\z@skip
4829     \@rightskip\@flushglue
4830     \bbl@endskip\@rightskip
4831     \parindent\z@
4832     \parfillskip\bbl@startskip}
4833   \def\raggedleft{%
4834     \let\@centercr
4835     \bbl@startskip\@flushglue
4836     \bbl@endskip\z@skip
4837     \parindent\z@
4838     \parfillskip\bbl@endskip}
4839 \fi
4840 \IfBabelLayout{lists}
4841   {\bbl@sreplace\list
4842     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4843     \def\bbl@listleftmargin{%
4844       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4845     \ifcase\bbl@engine
4846       \def\labelenumii{}\theenumii{}% pdftex doesn't reverse ()
4847       \def\p@enumiii{\p@enumii}\theenumii{}%
4848     \fi
4849     \bbl@sreplace\@verbatim
4850       {\leftskip\@totalleftmargin}%
4851       {\bbl@startskip\textwidth
4852         \advance\bbl@startskip-\linewidth}%
4853     \bbl@sreplace\@verbatim
4854       {\rightskip\z@skip}%
4855     {\bbl@endskip\z@skip}}%
4856   {}

```



```

4857 \IfBabelLayout{contents}
4858   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4859     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4860   {}
4861 \IfBabelLayout{columns}
4862   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4863     \def\bbl@outputbox#1{%
4864       \hb@xt@\textwidth{%
4865         \hskip\columnwidth
4866         \hfil
4867         {\normalcolor\vrule \@width\columnseprule}%
4868         \hfil
4869         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4870         \hskip-\textwidth
4871         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4872         \hskip\columnsep
4873         \hskip\columnwidth}}}%
4874   {}
4875 <<Footnote changes>>
4876 \IfBabelLayout{footnotes}%
4877   {\BabelFootnote\footnote\languagename{}{}}%
4878   \BabelFootnote\localfootnote\languagename{}{}}%
4879   \BabelFootnote\mainfootnote{}{}}%
4880   {}

```

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.

```

4881 \IfBabelLayout{counters*}%
4882   {\bbl@add\bbl@opt@layout{.counters.}%
4883     \AddToHook{shipout/before}{%
4884       \let\bbl@tempa\babelsublr
4885       \let\babelsublr\@firstofone
4886       \let\bbl@save@thepage\thepage
4887       \protected@edef\thepage{\thepage}%
4888       \let\babelsublr\bbl@tempa}%
4889     \AddToHook{shipout/after}{%
4890       \let\thepage\bbl@save@thepage}}}%
4891 \IfBabelLayout{counters}%
4892   {\let\bbl@latinarabic=\@arabic
4893     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4894     \let\bbl@asciroman=\@roman
4895     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4896     \let\bbl@asciiRoman=\@Roman
4897     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}%
4898 \fi % end if layout
4899 </xetex | texpet>

```

9.3 8-bit TeX

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

```

4900 <*texpet>
4901 \def\bbl@provide@extra#1{%
4902   % == auto-select encoding ==
4903   \ifx\bbl@encoding@select@off\empty\else
4904     \bbl@ifunset{\bbl@encoding@#1}%
4905     {\def\@elt##1{,##1,}%
4906       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4907       \count@\z@
4908       \bbl@foreach\bbl@tempe{%
4909         \def\bbl@tempd{##1}% Save last declared
4910         \advance\count@\@ne}%
4911       \ifnum\count@>\@ne
4912         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%

```

```

4913 \ifx\bbbl@tempa\relax \let\bbbl@tempa\@empty \fi
4914 \bbbl@replace\bbbl@tempa{ }{,}%
4915 \global\bbbl@csarg\let{encoding@#1}\@empty
4916 \bbbl@xin@{,\bbbl@tempd,}{,\bbbl@tempa,}%
4917 \ifin@%else % if main encoding included in ini, do nothing
4918 \let\bbbl@tempb\relax
4919 \bbbl@foreach\bbbl@tempa{%
4920 \ifx\bbbl@tempb\relax
4921 \bbbl@xin@{,##1,}{,\bbbl@tempe,}%
4922 \ifin@\def\bbbl@tempb{##1}\fi
4923 \fi}%
4924 \ifx\bbbl@tempb\relax\else
4925 \bbbl@exp{%
4926 \global\<\bbbl@add>\<\bbbl@preextras@#1>\<\bbbl@encoding@#1>}%
4927 \gdef\<\bbbl@encoding@#1>{%
4928 \\\babel@save\\f@encoding
4929 \\\bbbl@add\\originalTeX{\\selectfont}%
4930 \\\fontencoding{\bbbl@tempb}%
4931 \\\selectfont}}%
4932 \fi
4933 \fi
4934 \fi}%
4935 {}%
4936 \fi}
4937 \</texxet>

```

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, `\bbbl@hyphendata@<num>` exists (with the names of the files read).

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

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

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

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

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

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

```

4938 \*luatex>
4939 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4940 \bbbl@trace{Read language.dat}

```

```

4941 \ifx\bb@readstream\undefined
4942 \csname newread\endcsname\bb@readstream
4943 \fi
4944 \begingroup
4945 \toks@{}
4946 \count@ \z@ % 0=start, 1=0th, 2=normal
4947 \def\bb@process@line#1#2 #3 #4 {%
4948 \ifx=#1%
4949 \bb@process@synonym{#2}%
4950 \else
4951 \bb@process@language{#1#2}{#3}{#4}%
4952 \fi
4953 \ignorespaces}
4954 \def\bb@manylang{%
4955 \ifnum\bb@last>\@ne
4956 \bb@info{Non-standard hyphenation setup}%
4957 \fi
4958 \let\bb@manylang\relax}
4959 \def\bb@process@language#1#2#3{%
4960 \ifcase\count@
4961 \ifundefined{zth#1}{\count@\tw@}{\count@\@ne}%
4962 \or
4963 \count@\tw@
4964 \fi
4965 \ifnum\count@=\tw@
4966 \expandafter\addlanguage\csname l@#1\endcsname
4967 \language\allocationnumber
4968 \chardef\bb@last\allocationnumber
4969 \bb@manylang
4970 \let\bb@elt\relax
4971 \xdef\bb@languages{%
4972 \bb@languages\bb@elt{#1}{\the\language}{#2}{#3}}%
4973 \fi
4974 \the\toks@
4975 \toks@{}}
4976 \def\bb@process@synonym@aux#1#2{%
4977 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4978 \let\bb@elt\relax
4979 \xdef\bb@languages{%
4980 \bb@languages\bb@elt{#1}{#2}{}}}%
4981 \def\bb@process@synonym#1{%
4982 \ifcase\count@
4983 \toks@\expandafter{\the\toks@\relax\bb@process@synonym{#1}}%
4984 \or
4985 \@ifundefined{zth#1}{\bb@process@synonym@aux{#1}{0}}}%
4986 \else
4987 \bb@process@synonym@aux{#1}{\the\bb@last}%
4988 \fi}
4989 \ifx\bb@languages\undefined % Just a (sensible?) guess
4990 \chardef\l@english\z@
4991 \chardef\l@USenglish\z@
4992 \chardef\bb@last\z@
4993 \global\@namedef{bb@hyphendata@0}{{hyphen.tex}}
4994 \gdef\bb@languages{%
4995 \bb@elt{english}{0}{hyphen.tex}}%
4996 \bb@elt{USenglish}{0}{}}
4997 \else
4998 \global\let\bb@languages@format\bb@languages
4999 \def\bb@elt#1#2#3#4{% Remove all except language 0
5000 \ifnum#2>\z@\else
5001 \noexpand\bb@elt{#1}{#2}{#3}{#4}%
5002 \fi}%
5003 \xdef\bb@languages{\bb@languages}%

```

```

5004 \fi
5005 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
5006 \bbl@languages
5007 \openin\bbl@readstream=language.dat
5008 \ifeof\bbl@readstream
5009 \bbl@warning{I couldn't find language.dat. No additional\\%
5010 patterns loaded. Reported}%
5011 \else
5012 \loop
5013 \endlinechar\m@ne
5014 \read\bbl@readstream to \bbl@line
5015 \endlinechar\^^M
5016 \if T\ifeof\bbl@readstream F\fi T\relax
5017 \ifx\bbl@line\empty\else
5018 \edef\bbl@line{\bbl@line\space\space\space}%
5019 \expandafter\bbl@process@line\bbl@line\relax
5020 \fi
5021 \repeat
5022 \fi
5023 \closein\bbl@readstream
5024 \endgroup
5025 \bbl@trace{Macros for reading patterns files}
5026 \def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
5027 \ifx\babelcatcodetablenum\undefined
5028 \ifx\newcatcodetable\undefined
5029 \def\babelcatcodetablenum{5211}
5030 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5031 \else
5032 \newcatcodetable\babelcatcodetablenum
5033 \newcatcodetable\bbl@pattcodes
5034 \fi
5035 \else
5036 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5037 \fi
5038 \def\bbl@luapatterns#1#2{%
5039 \bbl@get@enc#1::\@@@
5040 \setbox\z@\hbox\bgroup
5041 \begingroup
5042 \savecatcodetable\babelcatcodetablenum\relax
5043 \initcatcodetable\bbl@pattcodes\relax
5044 \catcodetable\bbl@pattcodes\relax
5045 \catcode\#=6 \catcode\$\=3 \catcode\&\=4 \catcode\^=\=7
5046 \catcode\_=\=8 \catcode\{=\=1 \catcode\}\=2 \catcode\~=\=13
5047 \catcode\@=\=11 \catcode\^^I=\=10 \catcode\^^J=\=12
5048 \catcode\<=\=12 \catcode\>=\=12 \catcode\*=12 \catcode\.=12
5049 \catcode\-=12 \catcode\/=\=12 \catcode\[=\=12 \catcode\]=12
5050 \catcode\`=12 \catcode\'=\=12 \catcode\"=\=12
5051 \input #1\relax
5052 \catcodetable\babelcatcodetablenum\relax
5053 \endgroup
5054 \def\bbl@tempa{#2}%
5055 \ifx\bbl@tempa\empty\else
5056 \input #2\relax
5057 \fi
5058 \egroup}%
5059 \def\bbl@patterns@lua#1{%
5060 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5061 \csname l@#1\endcsname
5062 \edef\bbl@tempa{#1}%
5063 \else
5064 \csname l@#1:\f@encoding\endcsname
5065 \edef\bbl@tempa{#1:\f@encoding}%
5066 \fi\relax

```

```

5067 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5068 \@ifundefined{bbl@hyphendata@the\language}%
5069 {\def\bbl@elt##1##2##3##4{%
5070     \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:0T1...
5071     \def\bbl@tempb{##3}%
5072     \ifx\bbl@tempb\empty\else % if not a synonymous
5073     \def\bbl@tempc{##3}{##4}}%
5074     \fi
5075     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5076     \fi}%
5077 \bbl@languages
5078 \@ifundefined{bbl@hyphendata@the\language}%
5079 {\bbl@info{No hyphenation patterns were set for\%
5080     language '\bbl@tempa'. Reported}}%
5081 {\expandafter\expandafter\expandafter\bbl@luapatterns
5082     \csname bbl@hyphendata@the\language\endcsname}}}}
5083 \endinput\fi
5084 % Here ends \ifx\AddBabelHook\@undefined
5085 % A few lines are only read by hyphen.cfg
5086 \ifx\DisableBabelHook\@undefined
5087 \AddBabelHook{luatex}{everylanguage}{%
5088     \def\process@language##1##2##3{%
5089         \def\process@line####1####2 ####3 ####4 {}}}
5090 \AddBabelHook{luatex}{loadpatterns}{%
5091     \input #1\relax
5092     \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5093         {{#1}}}}
5094 \AddBabelHook{luatex}{loadexceptions}{%
5095     \input #1\relax
5096     \def\bbl@tempb##1##2{{##1}{#1}}%
5097     \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5098         {\expandafter\expandafter\expandafter\bbl@tempb
5099             \csname bbl@hyphendata@the\language\endcsname}}
5100 \endinput\fi
5101 % Here stops reading code for hyphen.cfg
5102 % The following is read the 2nd time it's loaded
5103 \begingroup % TODO - to a lua file
5104 \catcode`\%=12
5105 \catcode`\'=12
5106 \catcode`\%=12
5107 \catcode`\:=12
5108 \directlua{
5109     Babel = Babel or {}
5110     function Babel.bytes(line)
5111         return line:gsub(".",
5112             function (chr) return unicode.utf8.char(string.byte(chr)) end)
5113     end
5114     function Babel.begin_process_input()
5115         if luatexbase and luatexbase.add_to_callback then
5116             luatexbase.add_to_callback('process_input_buffer',
5117                 Babel.bytes, 'Babel.bytes')
5118         else
5119             Babel.callback = callback.find('process_input_buffer')
5120             callback.register('process_input_buffer', Babel.bytes)
5121         end
5122     end
5123     function Babel.end_process_input ()
5124         if luatexbase and luatexbase.remove_from_callback then
5125             luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
5126         else
5127             callback.register('process_input_buffer', Babel.callback)
5128         end
5129     end

```

```

5130 function Babel.addpatterns(pp, lg)
5131   local lg = lang.new(lg)
5132   local pats = lang.patterns(lg) or ''
5133   lang.clear_patterns(lg)
5134   for p in pp:gmatch('[^%s]+') do
5135     ss = ''
5136     for i in string.utfcharacters(p:gsub('%d', '')) do
5137       ss = ss .. '%d?' .. i
5138     end
5139     ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
5140     ss = ss:gsub('%.%%d%?$', '%%.')
5141     pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5142     if n == 0 then
5143       tex.sprint(
5144         [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5145         .. p .. [[]])
5146       pats = pats .. ' ' .. p
5147     else
5148       tex.sprint(
5149         [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5150         .. p .. [[]])
5151     end
5152   end
5153   lang.patterns(lg, pats)
5154 end
5155 Babel.characters = Babel.characters or {}
5156 Babel.ranges = Babel.ranges or {}
5157 function Babel.hlist_has_bidi(head)
5158   local has_bidi = false
5159   local ranges = Babel.ranges
5160   for item in node.traverse(head) do
5161     if item.id == node.id'glyph' then
5162       local itemchar = item.char
5163       local chardata = Babel.characters[itemchar]
5164       local dir = chardata and chardata.d or nil
5165       if not dir then
5166         for nn, et in ipairs(ranges) do
5167           if itemchar < et[1] then
5168             break
5169           elseif itemchar <= et[2] then
5170             dir = et[3]
5171             break
5172           end
5173         end
5174       end
5175       if dir and (dir == 'al' or dir == 'r') then
5176         has_bidi = true
5177       end
5178     end
5179   end
5180   return has_bidi
5181 end
5182 function Babel.set_chranges_b (script, chrng)
5183   if chrng == '' then return end
5184   texio.write('Replacing ' .. script .. ' script ranges')
5185   Babel.script_blocks[script] = {}
5186   for s, e in string.gmatch(chrng..' ', '(.-%.(-(.)%s)') do
5187     table.insert(
5188       Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5189   end
5190 end
5191 function Babel.discard_sublr(str)
5192   if str:find( [[\string\indexentry]] ) and

```

```

5193         str:find( [[\string\babelsublr]] ) then
5194         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5195                         function(m) return m:sub(2,-2) end )
5196     end
5197     return str
5198 end
5199 }
5200 \endgroup
5201 \ifx\newattribute\undefined\else
5202   \newattribute\bbl@attr@locale
5203   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5204   \AddBabelHook{luatex}{beforeextras}{%
5205     \setattribute\bbl@attr@locale\localeid}
5206 \fi
5207 \def\BabelStringsDefault{unicode}
5208 \let\luabbl@stop\relax
5209 \AddBabelHook{luatex}{encodedcommands}{%
5210   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5211   \ifx\bbl@tempa\bbl@tempb\else
5212     \directlua{Babel.begin_process_input()}%
5213     \def\luabbl@stop{%
5214       \directlua{Babel.end_process_input()}}%
5215   \fi}%
5216 \AddBabelHook{luatex}{stopcommands}{%
5217   \luabbl@stop
5218   \let\luabbl@stop\relax}
5219 \AddBabelHook{luatex}{patterns}{%
5220   \@ifundefined{bbl@hyphendata@the\language}%
5221     {\def\bbl@elt##1##2##3##4{%
5222       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5223       \def\bbl@tempb{##3}%
5224       \ifx\bbl@tempb\@empty\else % if not a synonymous
5225         \def\bbl@tempc{{##3}{##4}}%
5226       \fi
5227       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5228     \fi}%
5229   \bbl@languages
5230   \@ifundefined{bbl@hyphendata@the\language}%
5231     {\bbl@info{No hyphenation patterns were set for\\
5232       language '#2'. Reported}}%
5233     {\expandafter\expandafter\expandafter\bbl@luapatterns
5234       \csname bbl@hyphendata@the\language\endcsname}}}%
5235   \@ifundefined{bbl@patterns@}{}%
5236   \begingroup
5237     \bbl@xin@{, \number\language,}{, \bbl@pttnlist}%
5238     \ifin@else
5239       \ifx\bbl@patterns@\@empty\else
5240         \directlua{ Babel.addpatterns(
5241           [[\bbl@patterns@]], \number\language) }%
5242       \fi
5243       \@ifundefined{bbl@patterns@#1}%
5244       \@empty
5245       {\directlua{ Babel.addpatterns(
5246         [[\space\csname bbl@patterns@#1\endcsname]],
5247         \number\language) }}%
5248       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5249     \fi
5250   \endgroup}%
5251 \bbl@exp{%
5252   \bbl@ifunset{bbl@prehc@\languagename}{}%
5253   {\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}}%
5254   {\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.

```

5255 \onlypreamble\babelpatterns
5256 \AtEndOfPackage{%
5257   \newcommand\babelpatterns[2][\@empty]{%
5258     \ifx\bbl@patterns@relax
5259       \let\bbl@patterns@\@empty
5260     \fi
5261     \ifx\bbl@pttnlist\@empty\else
5262       \bbl@warning{%
5263         You must not intermingle \string\selectlanguage\space and\\%
5264         \string\babelpatterns\space or some patterns will not\\%
5265         be taken into account. Reported}%
5266       \fi
5267       \ifx\@empty#1%
5268         \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5269       \else
5270         \edef\bbl@tempb{\zap@space#1 \@empty}%
5271         \bbl@for\bbl@tempa\bbl@tempb{%
5272           \bbl@fixname\bbl@tempa
5273           \bbl@iflanguage\bbl@tempa{%
5274             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5275               \ifundefined\bbl@patterns@\bbl@tempa}%
5276             \@empty
5277             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5278             #2}}}%
5279       \fi}}

```

9.5 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`. Replace regular (ie, implicit) discretionary hyphens by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary hyphens are not touched. See Unicode UAX 14.

```

5280 % TODO - to a lua file
5281 \directlua{
5282   Babel = Babel or {}
5283   Babel.linebreaking = Babel.linebreaking or {}
5284   Babel.linebreaking.before = {}
5285   Babel.linebreaking.after = {}
5286   Babel.locale = {} % Free to use, indexed by \localeid
5287   function Babel.linebreaking.add_before(func, pos)
5288     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5289     if pos == nil then
5290       table.insert(Babel.linebreaking.before, func)
5291     else
5292       table.insert(Babel.linebreaking.before, pos, func)
5293     end
5294   end
5295   function Babel.linebreaking.add_after(func)
5296     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5297     table.insert(Babel.linebreaking.after, func)
5298   end
5299 }
5300 \def\bbl@intraspace#1 #2 #3\@@{%
5301   \directlua{
5302     Babel = Babel or {}
5303     Babel.intraspaces = Babel.intraspaces or {}
5304     Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5305       {b = #1, p = #2, m = #3}
5306     Babel.locale_props[\the\localeid].intraspace = %

```



```

5307     {b = #1, p = #2, m = #3}
5308   }}
5309 \def\bbl@intrapenalty#1\@@{%
5310   \directlua{
5311     Babel = Babel or {}
5312     Babel.intrapenalties = Babel.intrapenalties or {}
5313     Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5314     Babel.locale_props[\the\localeid].intrapenalty = #1
5315   }}
5316 \begingroup
5317 \catcode`\%=12
5318 \catcode`\^=14
5319 \catcode`\'=12
5320 \catcode`\~=12
5321 \gdef\bbl@seaintraspace{^
5322   \let\bbl@seaintraspace\relax
5323   \directlua{
5324     Babel = Babel or {}
5325     Babel.sea_enabled = true
5326     Babel.sea_ranges = Babel.sea_ranges or {}
5327     function Babel.set_chrngs (script, chrng)
5328       local c = 0
5329       for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5330         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5331         c = c + 1
5332       end
5333     end
5334     function Babel.sea_disc_to_space (head)
5335       local sea_ranges = Babel.sea_ranges
5336       local last_char = nil
5337       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5338       for item in node.traverse(head) do
5339         local i = item.id
5340         if i == node.id'glyph' then
5341           last_char = item
5342         elseif i == 7 and item.subtype == 3 and last_char
5343           and last_char.char > 0x0C99 then
5344           quad = font.getfont(last_char.font).size
5345           for lg, rg in pairs(sea_ranges) do
5346             if last_char.char > rg[1] and last_char.char < rg[2] then
5347               lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5348               local intraspace = Babel.intraspaces[lg]
5349               local intrapenalty = Babel.intrapenalties[lg]
5350               local n
5351               if intrapenalty ~= 0 then
5352                 n = node.new(14, 0)      ^% penalty
5353                 n.penalty = intrapenalty
5354                 node.insert_before(head, item, n)
5355               end
5356               n = node.new(12, 13)      ^% (glue, spaceskip)
5357               node.setglue(n, intraspace.b * quad,
5358                 intraspace.p * quad,
5359                 intraspace.m * quad)
5360               node.insert_before(head, item, n)
5361               node.remove(head, item)
5362             end
5363           end
5364         end
5365       end
5366     end
5367   }^^
5368   \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.

```
5369 \catcode`\%=14
5370 \gdef\bbl@cjkintraspacespace{%
5371   \let\bbl@cjkintraspacespace\relax
5372   \directlua{
5373     Babel = Babel or {}
5374     require('babel-data-cjk.lua')
5375     Babel.cjk_enabled = true
5376     function Babel.cjk_linebreak(head)
5377       local GLYPH = node.id'glyph'
5378       local last_char = nil
5379       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5380       local last_class = nil
5381       local last_lang = nil
5382
5383       for item in node.traverse(head) do
5384         if item.id == GLYPH then
5385
5386           local lang = item.lang
5387
5388           local LOCALE = node.get_attribute(item,
5389             Babel.attr_locale)
5390           local props = Babel.locale_props[LOCALE]
5391
5392           local class = Babel.cjk_class[item.char].c
5393
5394           if props.cjk_quotes and props.cjk_quotes[item.char] then
5395             class = props.cjk_quotes[item.char]
5396           end
5397
5398           if class == 'cp' then class = 'cl' end % )) as CL
5399           if class == 'id' then class = 'I' end
5400
5401           local br = 0
5402           if class and last_class and Babel.cjk_breaks[last_class][class] then
5403             br = Babel.cjk_breaks[last_class][class]
5404           end
5405
5406           if br == 1 and props.linebreak == 'c' and
5407             lang ~= \the\l@nohyphenation\space and
5408             last_lang ~= \the\l@nohyphenation then
5409             local intrapenalty = props.intrapenalty
5410             if intrapenalty ~= 0 then
5411               local n = node.new(14, 0)      % penalty
5412               n.penalty = intrapenalty
5413               node.insert_before(head, item, n)
5414             end
5415             local intraspacespace = props.intraspacespace
5416             local n = node.new(12, 13)      % (glue, spaceskip)
5417             node.setglue(n, intraspacespace.b * quad,
5418               intraspacespace.p * quad,
5419               intraspacespace.m * quad)
5420             node.insert_before(head, item, n)
5421           end
5422
5423           if font.getfont(item.font) then
```

```

5424         quad = font.getfont(item.font).size
5425     end
5426     last_class = class
5427     last_lang = lang
5428     else % if penalty, glue or anything else
5429         last_class = nil
5430     end
5431 end
5432 lang.hyphenate(head)
5433 end
5434 }%
5435 \bbl@lua hyphenate}
5436 \gdef\bbl@lua hyphenate{%
5437 \let\bbl@lua hyphenate\relax
5438 \directlua{
5439     luatexbase.add_to_callback('hyphenate',
5440     function (head, tail)
5441         if Babel.linebreaking.before then
5442             for k, func in ipairs(Babel.linebreaking.before) do
5443                 func(head)
5444             end
5445         end
5446         if Babel.cjk_enabled then
5447             Babel.cjk_linebreak(head)
5448         end
5449         lang.hyphenate(head)
5450         if Babel.linebreaking.after then
5451             for k, func in ipairs(Babel.linebreaking.after) do
5452                 func(head)
5453             end
5454         end
5455         if Babel.sea_enabled then
5456             Babel.sea_disc_to_space(head)
5457         end
5458     end,
5459     'Babel.hyphenate')
5460 }
5461 }
5462 \endgroup
5463 \def\bbl@provide@intraspace{%
5464 \bbl@ifunset\bbl@intsp@{language name}{}%
5465     {\expandafter\ifx\cscname\bbl@intsp@{language name}\endcsname\empty\else
5466     \bbl@xin@{c}{\bbl@cl{lnbrk}}}%
5467     \ifin@           % cjk
5468     \bbl@cjk intraspace
5469     \directlua{
5470         Babel = Babel or {}
5471         Babel.locale_props = Babel.locale_props or {}
5472         Babel.locale_props[\the\localeid].linebreak = 'c'
5473     }%
5474     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@@}%
5475     \ifx\bbl@KVP@intrapenalty\@nnil
5476     \bbl@intrapenalty0\@@
5477     \fi
5478 \else           % sea
5479     \bbl@seaintraspace
5480     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@@}%
5481     \directlua{
5482         Babel = Babel or {}
5483         Babel.sea_ranges = Babel.sea_ranges or {}
5484         Babel.set_chranges('\bbl@cl{sbcpr}',
5485                             '\bbl@cl{chrng}')
5486     }%

```

```

5487 \ifx\bbl@KVP@intrapenalty\@nnil
5488 \bbl@intrapenalty0\@@
5489 \fi
5490 \fi
5491 \fi
5492 \ifx\bbl@KVP@intrapenalty\@nnil\else
5493 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5494 \fi}}

```

9.7 Arabic justification

WIP. \bbl@arabicjust is executed with both elongated and kashida. This must be fine tuned. The attribute kashida is set by transforms with kashida-

```

5495 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5496 \def\bblar@chars{%
5497   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5498   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5499   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5500 \def\bblar@elongated{%
5501   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5502   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5503   0649,064A}
5504 \beginngroup
5505 \catcode\_ =11 \catcode`:=11
5506 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5507 \endgroup
5508 \gdef\bbl@arabicjust{%
5509 \let\bbl@arabicjust\relax
5510 \newattribute\bblar@kashida
5511 \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5512 \bblar@kashida=\z@
5513 \bbl@patchfont{\bbl@parsejalt}}%
5514 \directlua{
5515   Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5516   Babel.arabic.elong_map[\the\localeid] = {}
5517   luatexbase.add_to_callback('post_linebreak_filter',
5518     Babel.arabic.justify, 'Babel.arabic.justify')
5519   luatexbase.add_to_callback('hpack_filter',
5520     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5521 }}%

```

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

```

5522 \def\bblar@fetchjalt#1#2#3#4{%
5523 \bbl@exp{\bbl@foreach{#1}}{%
5524 \bbl@ifunset\bblar@JE@##1}%
5525 {\setbox\z@\hbox{\textrm{TRT ^^^200d\char"##1#2}}%
5526 {\setbox\z@\hbox{\textrm{TRT ^^^200d\char"@nameuse\bblar@JE@##1#2}}}%
5527 \directlua{%
5528   local last = nil
5529   for item in node.traverse(tex.box[0].head) do
5530     if item.id == node.id'glyph' and item.char > 0x600 and
5531     not (item.char == 0x200D) then
5532       last = item
5533     end
5534   end
5535   Babel.arabic.#3['##1#4'] = last.char
5536 }}%

```

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

```

5537 \gdef\bbl@parsejalt{%
5538 \ifx\addfontfeature\undefined\else
5539 \bbl@xin@{/e}{\bbl@cl{\lnbrk}}%

```

```

5540 \ifin@
5541 \directlua{%
5542   if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5543     Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5544     tex.print([\string\csname\space bbl@parsejalti\endcsname])
5545   end
5546 }%
5547 \fi
5548 \fi}
5549 \gdef\bbl@parsejalti{%
5550 \begingroup
5551 \let\bbl@parsejalt\relax % To avoid infinite loop
5552 \edef\bbl@tempb{\fontid\font}%
5553 \bblar@nofswarn
5554 \bblar@fetchjalt\bblar@elongated{}{from}{}%
5555 \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5556 \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5557 \addfontfeature{RawFeature+=jalt}%
5558 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5559 \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5560 \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5561 \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5562 \directlua{%
5563   for k, v in pairs(Babel.arabic.from) do
5564     if Babel.arabic.dest[k] and
5565       not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5566       Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5567       [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5568     end
5569   end
5570 }%
5571 \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5572 \begingroup
5573 \catcode`#=11
5574 \catcode`~ =11
5575 \directlua{
5576
5577 Babel.arabic = Babel.arabic or {}
5578 Babel.arabic.from = {}
5579 Babel.arabic.dest = {}
5580 Babel.arabic.justify_factor = 0.95
5581 Babel.arabic.justify_enabled = true
5582 Babel.arabic.kashida_limit = -1
5583
5584 function Babel.arabic.justify(head)
5585   if not Babel.arabic.justify_enabled then return head end
5586   for line in node.traverse_id(node.id'hlist', head) do
5587     Babel.arabic.justify_hlist(head, line)
5588   end
5589   return head
5590 end
5591
5592 function Babel.arabic.justify_hbox(head, gc, size, pack)
5593   local has_inf = false
5594   if Babel.arabic.justify_enabled and pack == 'exactly' then
5595     for n in node.traverse_id(12, head) do
5596       if n.stretch_order > 0 then has_inf = true end
5597     end
5598     if not has_inf then
5599       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5600     end

```

```

5601 end
5602 return head
5603 end
5604
5605 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5606     local d, new
5607     local k_list, k_item, pos_inline
5608     local width, width_new, full, k_curr, wt_pos, goal, shift
5609     local subst_done = false
5610     local elong_map = Babel.arabic.elong_map
5611     local cnt
5612     local last_line
5613     local GLYPH = node.id'glyph'
5614     local KASHIDA = Babel.attr_kashida
5615     local LOCALE = Babel.attr_locale
5616
5617     if line == nil then
5618         line = {}
5619         line.glue_sign = 1
5620         line.glue_order = 0
5621         line.head = head
5622         line.shift = 0
5623         line.width = size
5624     end
5625
5626     % Exclude last line. todo. But-- it discards one-word lines, too!
5627     % ? Look for glue = 12:15
5628     if (line.glue_sign == 1 and line.glue_order == 0) then
5629         elongs = {} % Stores elongated candidates of each line
5630         k_list = {} % And all letters with kashida
5631         pos_inline = 0 % Not yet used
5632
5633         for n in node.traverse_id(GLYPH, line.head) do
5634             pos_inline = pos_inline + 1 % To find where it is. Not used.
5635
5636             % Elongated glyphs
5637             if elong_map then
5638                 local locale = node.get_attribute(n, LOCALE)
5639                 if elong_map[locale] and elong_map[locale][n.font] and
5640                     elong_map[locale][n.font][n.char] then
5641                     table.insert(elongs, {node = n, locale = locale} )
5642                     node.set_attribute(n.prev, KASHIDA, 0)
5643                 end
5644             end
5645
5646             % Tatwil
5647             if Babel.kashida_wts then
5648                 local k_wt = node.get_attribute(n, KASHIDA)
5649                 if k_wt > 0 then % todo. parameter for multi inserts
5650                     table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5651                 end
5652             end
5653
5654         end % of node.traverse_id
5655
5656         if #elongs == 0 and #k_list == 0 then goto next_line end
5657         full = line.width
5658         shift = line.shift
5659         goal = full * Babel.arabic.justify_factor % A bit crude
5660         width = node.dimensions(line.head) % The 'natural' width
5661
5662         % == Elongated ==
5663         % Original idea taken from 'chickenize'

```

```

5664 while (#elongs > 0 and width < goal) do
5665     subst_done = true
5666     local x = #elongs
5667     local curr = elongs[x].node
5668     local oldchar = curr.char
5669     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5670     width = node.dimensions(line.head) % Check if the line is too wide
5671     % Substitute back if the line would be too wide and break:
5672     if width > goal then
5673         curr.char = oldchar
5674         break
5675     end
5676     % If continue, pop the just substituted node from the list:
5677     table.remove(elongs, x)
5678 end
5679
5680 % == Tatwil ==
5681 if #k_list == 0 then goto next_line end
5682
5683 width = node.dimensions(line.head) % The 'natural' width
5684 k_curr = #k_list % Traverse backwards, from the end
5685 wt_pos = 1
5686
5687 while width < goal do
5688     subst_done = true
5689     k_item = k_list[k_curr].node
5690     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5691         d = node.copy(k_item)
5692         d.char = 0x0640
5693         d.yoffset = 0
5694         d.xoffset = 0
5695         line.head, new = node.insert_after(line.head, k_item, d)
5696         width_new = node.dimensions(line.head)
5697         if width > goal or width == width_new then
5698             node.remove(line.head, new) % Better compute before
5699             break
5700         end
5701         if Babel.fix_diacr then
5702             Babel.fix_diacr(k_item.next)
5703         end
5704         width = width_new
5705     end
5706     if k_curr == 1 then
5707         k_curr = #k_list
5708         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5709     else
5710         k_curr = k_curr - 1
5711     end
5712 end
5713
5714 % Limit the number of tatweel by removing them. Not very efficient,
5715 % but it does the job in a quite predictable way.
5716 if Babel.arabic.kashida_limit > -1 then
5717     cnt = 0
5718     for n in node.traverse_id(GLYPH, line.head) do
5719         if n.char == 0x0640 then
5720             cnt = cnt + 1
5721             if cnt > Babel.arabic.kashida_limit then
5722                 node.remove(line.head, n)
5723             end
5724         else
5725             cnt = 0
5726         end
5727     end
5728 end

```

```

5727     end
5728 end
5729
5730 ::next_line::
5731
5732 % Must take into account marks and ins, see luatex manual.
5733 % Have to be executed only if there are changes. Investigate
5734 % what's going on exactly.
5735 if subst_done and not gc then
5736     d = node.hpack(line.head, full, 'exactly')
5737     d.shift = shift
5738     node.insert_before(head, line, d)
5739     node.remove(head, line)
5740 end
5741 end % if process line
5742 end
5743 }
5744 \endgroup
5745 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

9.8 Common stuff

```

5746 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5747 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5748 \DisableBabelHook{babel-fontspec}
5749 <<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 from 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.

```

5750 % TODO - to a lua file
5751 \directlua{
5752 Babel.script_blocks = {
5753   ['dflt'] = {},
5754   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5755               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFD}, {0x1EE0, 0x1EEF}},
5756   ['Armn'] = {{0x0530, 0x058F}},
5757   ['Beng'] = {{0x0980, 0x09FF}},
5758   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5759   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5760   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5761                {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5762   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5763   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5764                {0xAB00, 0xAB2F}},
5765   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5766   % Don't follow strictly Unicode, which places some Coptic letters in
5767   % the 'Greek and Coptic' block
5768   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5769   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5770               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5771               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5772               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5773               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5774               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5775   ['Hebr'] = {{0x0590, 0x05FF}},
5776   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF}},

```



```

5777         {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5778 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5779 ['Knda'] = {{0x0C80, 0x0CFF}},
5780 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5781             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5782             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5783 ['Lao'] = {{0x0E80, 0x0EFF}},
5784 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5785             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5786             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5787 ['Mahj'] = {{0x1150, 0x117F}},
5788 ['Mlym'] = {{0x0D00, 0x0D7F}},
5789 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5790 ['Orya'] = {{0x0B00, 0x0B7F}},
5791 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x11E0, 0x11FF}},
5792 ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5793 ['Taml'] = {{0x0B80, 0x0BFF}},
5794 ['Telu'] = {{0x0C00, 0x0C7F}},
5795 ['Tfng'] = {{0x2D30, 0x2D7F}},
5796 ['Thai'] = {{0x0E00, 0x0E7F}},
5797 ['Tibt'] = {{0x0F00, 0x0FFF}},
5798 ['Vaii'] = {{0xA500, 0xA63F}},
5799 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5800 }
5801
5802 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5803 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5804 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5805
5806 function Babel.locale_map(head)
5807   if not Babel.locale_mapped then return head end
5808
5809   local LOCALE = Babel.attr_locale
5810   local GLYPH = node.id('glyph')
5811   local inmath = false
5812   local toloc_save
5813   for item in node.traverse(head) do
5814     local toloc
5815     if not inmath and item.id == GLYPH then
5816       % Optimization: build a table with the chars found
5817       if Babel.chr_to_loc[item.char] then
5818         toloc = Babel.chr_to_loc[item.char]
5819       else
5820         for lc, maps in pairs(Babel.loc_to_scr) do
5821           for _, rg in pairs(maps) do
5822             if item.char >= rg[1] and item.char <= rg[2] then
5823               Babel.chr_to_loc[item.char] = lc
5824               toloc = lc
5825               break
5826             end
5827           end
5828         end
5829       end
5830       % Now, take action, but treat composite chars in a different
5831       % fashion, because they 'inherit' the previous locale. Not yet
5832       % optimized.
5833       if not toloc and
5834         (item.char >= 0x0300 and item.char <= 0x036F) or
5835         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5836         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5837         toloc = toloc_save
5838       end
5839       if toloc and Babel.locale_props[toloc] and

```

```

5840         Babel.locale_props[toloc].letters and
5841         tex.getcatcode(item.char) \string~= 11 then
5842         toloc = nil
5843     end
5844     if toloc and toloc > -1 then
5845         if Babel.locale_props[toloc].lg then
5846             item.lang = Babel.locale_props[toloc].lg
5847             node.set_attribute(item, LOCALE, toloc)
5848         end
5849         if Babel.locale_props[toloc]['/'..item.font] then
5850             item.font = Babel.locale_props[toloc]['/'..item.font]
5851         end
5852         toloc_save = toloc
5853     end
5854     elseif not inmath and item.id == 7 then % Apply recursively
5855         item.replace = item.replace and Babel.locale_map(item.replace)
5856         item.pre      = item.pre and Babel.locale_map(item.pre)
5857         item.post      = item.post and Babel.locale_map(item.post)
5858     elseif item.id == node.id'math' then
5859         inmath = (item.subtype == 0)
5860     end
5861 end
5862 return head
5863 end
5864 }

```

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

```

5865 \newcommand\babelcharproperty[1]{%
5866   \count@=#1\relax
5867   \ifvmode
5868     \expandafter\bbl@chprop
5869   \else
5870     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5871               vertical mode (preamble or between paragraphs)}%
5872     {See the manual for futher info}%
5873   \fi}
5874 \newcommand\bbl@chprop[3][\the\count@]{%
5875   \@tempcnta=#1\relax
5876   \bbl@ifunset{\bbl@chprop@#2}%
5877   {\bbl@error{No property named '#2'. Allowed values are\\%
5878               direction (bc), mirror (bmg), and linebreak (lb)}%
5879   {See the manual for futher info}}%
5880   {}%
5881   \loop
5882     \bbl@cs{chprop@#2}{#3}%
5883     \ifnum\count@<\@tempcnta
5884       \advance\count@\@ne
5885     \repeat}
5886 \def\bbl@chprop@direction#1{%
5887   \directlua{
5888     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5889     Babel.characters[\the\count@]['d'] = '#1'
5890   }}
5891 \let\bbl@chprop@bc\bbl@chprop@direction
5892 \def\bbl@chprop@mirror#1{%
5893   \directlua{
5894     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5895     Babel.characters[\the\count@]['m'] = '\number#1'
5896   }}
5897 \let\bbl@chprop@bmg\bbl@chprop@mirror
5898 \def\bbl@chprop@linebreak#1{%
5899   \directlua{

```

```

5900   Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5901   Babel.cjk_characters[\the\count@]['c'] = '#1'
5902 }
5903 \let\bbl@chprop@lb\bbl@chprop@linebreak
5904 \def\bbl@chprop@locale#1{%
5905   \directlua{
5906     Babel.chr_to_loc = Babel.chr_to_loc or {}
5907     Babel.chr_to_loc[\the\count@] =
5908       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5909   }

```

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.

```

5910 \directlua{
5911   Babel.nohyphenation = \the\l@nohyphenation
5912 }

```

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 $\text{Babel.capt_map}(m[1],1)$ 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).

```

5913 \begingroup
5914 \catcode`\~ = 12
5915 \catcode`\% = 12
5916 \catcode`\& = 14
5917 \catcode`\| = 12
5918 \gdef\babelprehyphenation{%&
5919   \ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
5920 \gdef\babelposthyphenation{%&
5921   \ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
5922 \gdef\bbl@settransform#1[#2]#3#4#5{%&
5923   \ifcase#1
5924     \bbl@activateprehyphen
5925   \or
5926     \bbl@activateposthyphen
5927   \fi
5928 \begingroup
5929   \def\babeltempa{\bbl@add@list\babeltempb}%&
5930   \let\babeltempb@empty
5931   \def\bbl@tempa{#5}%&
5932   \bbl@replace\bbl@tempa{,}{ ,}%& TODO. Ugly trick to preserve {}
5933   \expandafter\bbl@foreach\expandafter{\bbl@tempa}{%&
5934     \bbl@ifsamestring{##1}{remove}%&
5935     {\bbl@add@list\babeltempb{nil}}}%&
5936     {\directlua{
5937       local rep = {[##1]}
5938       rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5939       rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5940       rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5941       if #1 == 0 or #1 == 2 then
5942         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5943           'space = { ' .. '%2, %3, %4' .. ' }')
5944         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5945           'spacefactor = { ' .. '%2, %3, %4' .. ' }')
5946         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5947       else
5948         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5949         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5950         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)

```

```

5951         end
5952         tex.print([[string\babeltempa{[] .. rep .. [[]]])
5953     }]}&%
5954 \bbl@foreach\babeltempb{&%
5955     \bbl@forkv{##1}{&%
5956         \in{,###1,}{,nil,step,data,remove,insert,string,no,pre,&%
5957             no,post,penalty,kashida,space,spacefactor,}&%
5958         \ifin@%else
5959             \bbl@error
5960             {Bad option '###1' in a transform.\\&%
5961                 I'll ignore it but expect more errors}&%
5962             {See the manual for further info.}&%
5963         \fi}}&%
5964 \let\bbl@kv@attribute\relax
5965 \let\bbl@kv@label\relax
5966 \let\bbl@kv@fonts\@empty
5967 \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5968 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
5969 \ifx\bbl@kv@attribute\relax
5970     \ifx\bbl@kv@label\relax\else
5971         \bbl@exp{\bbl@trim@def\bbl@kv@fonts{\bbl@kv@fonts}}&%
5972         \bbl@replace\bbl@kv@fonts{ }{,}&%
5973         \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
5974         \count@\z@
5975         \def\bbl@elt##1##2##3{&%
5976             \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
5977             {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5978                 {\count@\@ne}&%
5979                 {\bbl@error
5980                     {Transforms cannot be re-assigned to different\\&%
5981                         fonts. The conflict is in '\bbl@kv@label'.\\&%
5982                         Apply the same fonts or use a different label}&%
5983                     {See the manual for further details.}}}&%
5984             {}}&%
5985         \bbl@transfont@list
5986         \ifnum\count@=\z@
5987             \bbl@exp{\global\bbl@add\bbl@transfont@list
5988                 {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
5989         \fi
5990         \bbl@ifunset{\bbl@kv@attribute}&%
5991         {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
5992         {}&%
5993         \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
5994     \fi
5995 \else
5996     \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5997 \fi
5998 \directlua{
5999     local lbkr = Babel.linebreaking.replacements[#1]
6000     local u = unicode.utf8
6001     local id, attr, label
6002     if #1 == 0 then
6003         id = \the\csname bbl@id@#3\endcsname\space
6004     else
6005         id = \the\csname l@#3\endcsname\space
6006     end
6007     \ifx\bbl@kv@attribute\relax
6008         attr = -1
6009     \else
6010         attr = luatexbase.registernumber'\bbl@kv@attribute'
6011     \fi
6012     \ifx\bbl@kv@label\relax\else &% Same refs:
6013         label = [==[\bbl@kv@label]==]

```

```

6014 \fi
6015 &% Convert pattern:
6016 local patt = string.gsub([==[#4]==], '%s', '')
6017 if #1 == 0 then
6018     patt = string.gsub(patt, '|', ' ')
6019 end
6020 if not u.find(patt, '()', nil, true) then
6021     patt = '()' .. patt .. '()'
6022 end
6023 if #1 == 1 then
6024     patt = string.gsub(patt, '%(%)^', '^()')
6025     patt = string.gsub(patt, '%$%(%)', '()$')
6026 end
6027 patt = u.gsub(patt, '{(.)}',
6028     function (n)
6029         return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6030     end)
6031 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6032     function (n)
6033         return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6034     end)
6035 lbkr[id] = lbkr[id] or {}
6036 table.insert(lbkr[id],
6037     { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6038 }&%
6039 \endgroup}
6040 \endgroup
6041 \let\bbl@transfont@list@empty
6042 \def\bbl@settransfont{%
6043     \global\let\bbl@settransfont\relax % Execute only once
6044     \gdef\bbl@transfont{%
6045         \def\bbl@elt####1####2####3{%
6046             \bbl@ifblank{####3}%
6047                 {\count@tw@}% Do nothing if no fonts
6048             {\count@z@
6049                 \bbl@vforeach{####3}{%
6050                     \def\bbl@tempd{#####1}%
6051                     \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6052                     \ifx\bbl@tempd\bbl@tempe
6053                         \count@ne
6054                     \else\ifx\bbl@tempd\bbl@transfam
6055                         \count@ne
6056                     \fi\fi}%
6057                 \ifcase\count@
6058                     \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
6059                 \or
6060                     \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
6061                 \fi}%
6062                 \bbl@transfont@list}%
6063     \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6064     \gdef\bbl@transfam{-unknown-}%
6065     \bbl@foreach\bbl@font@fams{%
6066         \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6067         \bbl@ifsamestring{\@nameuse{##1default}}{\familydefault
6068             {\xdef\bbl@transfam{##1}}%
6069         {}}}
6070 \DeclareRobustCommand\enablelocaletransform[1]{%
6071     \bbl@ifunset{\bbl@ATR@#1@language @}%
6072     {\bbl@error
6073         {'#1' for '\language' cannot be enabled.\\%
6074         Maybe there is a typo or it's a font-dependent transform}%
6075     {See the manual for further details.}}%
6076     {\bbl@csarg\setattribute{ATR@#1@language @}\@ne}}

```

```

6077 \DeclareRobustCommand\disablelocaletransform[1]{%
6078   \bbl@ifunset{\bbl@ATR@#1@\language@}{%
6079     {\bbl@error
6080       {'#1' for '\language@' cannot be disabled.\\%
6081         Maybe there is a typo or it's a font-dependent transform}%
6082       {See the manual for further details.}}%
6083     {\bbl@csarg\unsetattribute{ATR@#1@\language@}{}}
6084 \def\bbl@activateposthyphen{%
6085   \let\bbl@activateposthyphen\relax
6086   \directlua{
6087     require('babel-transforms.lua')
6088     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6089   }}
6090 \def\bbl@activateprehyphen{%
6091   \let\bbl@activateprehyphen\relax
6092   \directlua{
6093     require('babel-transforms.lua')
6094     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6095   }}

```

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

```

6096 \newcommand\localeprehyphenation[1]{%
6097   \directlua{ Babel.string_prehyphenation([=#1]=), \the\localeid) }}

```

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.

```

6098 \def\bbl@activate@preotf{%
6099   \let\bbl@activate@preotf\relax % only once
6100   \directlua{
6101     Babel = Babel or {}
6102     %
6103     function Babel.pre_otfload_v(head)
6104       if Babel.numbers and Babel.digits_mapped then
6105         head = Babel.numbers(head)
6106       end
6107       if Babel.bidi_enabled then
6108         head = Babel.bidi(head, false, dir)
6109       end
6110       return head
6111     end
6112     %
6113     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6114       if Babel.numbers and Babel.digits_mapped then
6115         head = Babel.numbers(head)
6116       end
6117       if Babel.bidi_enabled then
6118         head = Babel.bidi(head, false, dir)
6119       end
6120       return head
6121     end
6122     %
6123     luatexbase.add_to_callback('pre_linebreak_filter',
6124       Babel.pre_otfload_v,
6125       'Babel.pre_otfload_v',
6126     luatexbase.priority_in_callback('pre_linebreak_filter',
6127       'luaotfload.node_processor') or nil)

```

```

6128 %
6129 luatexbase.add_to_callback('hpack_filter',
6130     Babel.pre_otfload_h,
6131     'Babel.pre_otfload_h',
6132     luatexbase.priority_in_callback('hpack_filter',
6133     'luaotfload.node_processor') or nil)
6134 }}

```

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=`.

```

6135 \breakafterdirmode=1
6136 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6137   \let\bbl@beforeforeign\leavevmode
6138   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6139   \RequirePackage{luatexbase}
6140   \bbl@activate@preotf
6141   \directlua{
6142     require('babel-data-bidi.lua')
6143     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6144       require('babel-bidi-basic.lua')
6145     \or
6146       require('babel-bidi-basic-r.lua')
6147     \fi}
6148   \newattribute\bbl@attr@dir
6149   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6150   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6151 \fi
6152 \chardef\bbl@thetextdir\z@
6153 \chardef\bbl@thepardir\z@
6154 \def\bbl@getluadir#1{%
6155   \directlua{
6156     if tex.#1dir == 'TLT' then
6157       tex.sprint('0')
6158     elseif tex.#1dir == 'TRT' then
6159       tex.sprint('1')
6160     end}}
6161 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6162   \ifcase#3\relax
6163     \ifcase\bbl@getluadir{#1}\relax\else
6164       #2 TLT\relax
6165     \fi
6166   \else
6167     \ifcase\bbl@getluadir{#1}\relax
6168       #2 TRT\relax
6169     \fi
6170   \fi}
6171 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6172 \def\bbl@thedir{0}
6173 \def\bbl@textdir#1{%
6174   \bbl@setluadir{text}\textdir{#1}%
6175   \chardef\bbl@thetextdir#1\relax
6176   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6177   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6178 \def\bbl@pardir#1{% Used twice
6179   \bbl@setluadir{par}\pardir{#1}%
6180   \chardef\bbl@thepardir#1\relax}
6181 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6182 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6183 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once
6184 \ifnum\bbl@bidimode>\z@ % Any bidi=

```

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.

```

6185 \def\bbl@insidemath{0}%
6186 \def\bbl@everymath{\def\bbl@insidemath{1}}
6187 \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6188 \frozen@everymath\expandafter{%
6189   \expandafter\bbl@everymath\the\frozen@everymath}
6190 \frozen@everydisplay\expandafter{%
6191   \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6192 \AtBeginDocument{
6193   \directlua{
6194     function Babel.math_box_dir(head)
6195       if not (token.get_macro('bbl@insidemath') == '0') then
6196         if Babel.hlist_has_bidi(head) then
6197           local d = node.new(node.id'dir')
6198           d.dir = '+TRT'
6199           node.insert_before(head, node.has_glyph(head), d)
6200           for item in node.traverse(head) do
6201             node.set_attribute(item,
6202               Babel.attr_dir, token.get_macro('bbl@thedir'))
6203           end
6204         end
6205       end
6206       return head
6207     end
6208     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6209       "Babel.math_box_dir", 0)
6210   }}%
6211 \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.

```

6212 \bbl@trace{Redefinitions for bidi layout}
6213 %
6214 <<(*More package options)>> ≡
6215 \chardef\bbl@eqnpos\z@
6216 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6217 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6218 <</More package options>>
6219 %
6220 \ifnum\bbl@bidimode>\z@ % Any bidi=
6221   \matheqdirmode\@ne % A luatex primitive
6222   \let\bbl@eqnudir\relax
6223   \def\bbl@eqdel{()}
6224   \def\bbl@eqnum{%
6225     {\normalfont\normalcolor

```



```

6226 \expandafter\@firstoftwo\bbel@eqdel
6227 \theequation
6228 \expandafter\@secondoftwo\bbel@eqdel}}
6229 \def\bbel@puteqno#1{\eqno\hbox{#1}}
6230 \def\bbel@putleqno#1{\leqno\hbox{#1}}
6231 \def\bbel@eqno@flip#1{%
6232 \ifdim\predisplaysize=-\maxdimen
6233 \eqno
6234 \hbext@.01pt{\hbext@\displaywidth{\hss{#1}}\hss}%
6235 \else
6236 \leqno\hbox{#1}%
6237 \fi}
6238 \def\bbel@eqno@flip#1{%
6239 \ifdim\predisplaysize=-\maxdimen
6240 \leqno
6241 \hbext@.01pt{\hss\hbext@\displaywidth{#1}\hss}%
6242 \else
6243 \eqno\hbox{#1}%
6244 \fi}
6245 \AtBeginDocument{%
6246 \ifx\bbel@noamsmath\relax\else
6247 \ifx\maketag@@@undefined % Normal equation, eqnarray
6248 \AddToHook{env/equation/begin}{%
6249 \ifnum\bbel@thetextdir>\z@
6250 \def\bbel@mathboxdir{\def\bbel@insidemath{1}}%
6251 \let\@eqnnum\bbel@eqnum
6252 \edef\bbel@eqnodir{\noexpand\bbel@textdir{\the\bbel@thetextdir}}%
6253 \chardef\bbel@thetextdir\z@
6254 \bbel@add\normalfont{\bbel@eqnodir}%
6255 \ifcase\bbel@eqnpos
6256 \let\bbel@puteqno\bbel@eqno@flip
6257 \or
6258 \let\bbel@puteqno\bbel@eqno@flip
6259 \fi
6260 \fi}%
6261 \ifnum\bbel@eqnpos=\tw@\else
6262 \def\endequation{\bbel@puteqno{\@eqnnum}$$\@ignoretrue}%
6263 \fi
6264 \AddToHook{env/eqnarray/begin}{%
6265 \ifnum\bbel@thetextdir>\z@
6266 \def\bbel@mathboxdir{\def\bbel@insidemath{1}}%
6267 \edef\bbel@eqnodir{\noexpand\bbel@textdir{\the\bbel@thetextdir}}%
6268 \chardef\bbel@thetextdir\z@
6269 \bbel@add\normalfont{\bbel@eqnodir}%
6270 \ifnum\bbel@eqnpos=\@ne
6271 \def\@eqnnum{%
6272 \setbox\z@\hbox{\bbel@eqnum}%
6273 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6274 \else
6275 \let\@eqnnum\bbel@eqnum
6276 \fi
6277 \fi}
6278 % Hack. YA luatex bug?:
6279 \expandafter\bbel@sreplace\csname] \endcsname{${}\{\eqno\kern.001pt${}}%
6280 \else % amstex
6281 \bbel@exp{% Hack to hide maybe undefined conditionals:
6282 \chardef\bbel@eqnpos=0%
6283 \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6284 \ifnum\bbel@eqnpos=\@ne
6285 \let\bbel@ams@lap\hbox
6286 \else
6287 \let\bbel@ams@lap\llap
6288 \fi

```

```

6289 \ExplSyntaxOn % Required by \bbl@sreplace with \intertext@
6290 \bbl@sreplace\intertext@{\normalbaselines}%
6291 {\normalbaselines
6292 \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6293 \ExplSyntaxOff
6294 \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6295 \ifx\bbl@ams@lap\hbox % leqno
6296 \def\bbl@ams@flip#1{%
6297 \hbox to 0.01pt{\hss\hbox to\displaywidth{#{1}\hss}}}%
6298 \else % eqno
6299 \def\bbl@ams@flip#1{%
6300 \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}\hss}}}%
6301 \fi
6302 \def\bbl@ams@preset#1{%
6303 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6304 \ifnum\bbl@thetextdir>\z@
6305 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6306 \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6307 \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6308 \fi}%
6309 \ifnum\bbl@eqnpos=\tw@ \else
6310 \def\bbl@ams@equation{%
6311 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6312 \ifnum\bbl@thetextdir>\z@
6313 \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6314 \chardef\bbl@thetextdir\z@
6315 \bbl@add\normalfont{\bbl@eqnodir}%
6316 \ifcase\bbl@eqnpos
6317 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6318 \or
6319 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6320 \fi
6321 \fi}%
6322 \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6323 \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6324 \fi
6325 \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6326 \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6327 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6328 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6329 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6330 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6331 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6332 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6333 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6334 % Hackish, for proper alignment. Don't ask me why it works!:
6335 \bbl@exp{% Avoid a 'visible' conditional
6336 \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6337 \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{\<fi>}}%
6338 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6339 \AddToHook{env/split/before}{%
6340 \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6341 \ifnum\bbl@thetextdir>\z@
6342 \bbl@ifsamestring\@currentenv{equation}%
6343 {\ifx\bbl@ams@lap\hbox % leqno
6344 \def\bbl@ams@flip#1{%
6345 \hbox to 0.01pt{\hbox to\displaywidth{#{1}\hss}\hss}}%
6346 \else
6347 \def\bbl@ams@flip#1{%
6348 \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6349 \fi}%
6350 }%
6351 \fi}%

```

```

6352 \fi\fi}
6353 \fi
6354 \def\bbl@provide@extra#1{%
6355 % == Counters: mapdigits ==
6356 % Native digits
6357 \ifx\bbl@KVP@mapdigits\@nnil\else
6358 \bbl@ifunset\bbl@dgnat@{language\name}{}%
6359 {\RequirePackage{luatexbase}%
6360 \bbl@activate@preotf
6361 \directlua{
6362 Babel = Babel or {} %%% -> presets in luababel
6363 Babel.digits_mapped = true
6364 Babel.digits = Babel.digits or {}
6365 Babel.digits[\the\localeid] =
6366 table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6367 if not Babel.numbers then
6368 function Babel.numbers(head)
6369 local LOCALE = Babel.attr_locale
6370 local GLYPH = node.id'glyph'
6371 local inmath = false
6372 for item in node.traverse(head) do
6373 if not inmath and item.id == GLYPH then
6374 local temp = node.get_attribute(item, LOCALE)
6375 if Babel.digits[temp] then
6376 local chr = item.char
6377 if chr > 47 and chr < 58 then
6378 item.char = Babel.digits[temp][chr-47]
6379 end
6380 end
6381 elseif item.id == node.id'math' then
6382 inmath = (item.subtype == 0)
6383 end
6384 end
6385 return head
6386 end
6387 end
6388 }}%
6389 \fi
6390 % == transforms ==
6391 \ifx\bbl@KVP@transforms\@nnil\else
6392 \def\bbl@elt##1##2##3{%
6393 \in@{${transforms.}}{##1}%
6394 \ifin@
6395 \def\bbl@tempa{##1}%
6396 \bbl@replace\bbl@tempa{transforms.}{}%
6397 \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6398 \fi}%
6399 \csname bbl@inidata@{language\name}\endcsname
6400 \bbl@release@transforms\relax % \relax closes the last item.
6401 \fi}
6402 % Start tabular here:
6403 \def\localerestoredirs{%
6404 \ifcase\bbl@thetextdir
6405 \ifnum\textdirection=\z@\else\textdir TLT\fi
6406 \else
6407 \ifnum\textdirection=\@ne\else\textdir TRT\fi
6408 \fi
6409 \ifcase\bbl@thepardir
6410 \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6411 \else
6412 \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6413 \fi}
6414 \IfBabelLayout{tabular}%

```

```

6415 {\chardef\bb@tabular@mode\tw@}% All RTL
6416 {\IfBabelLayout{notabular}%
6417   {\chardef\bb@tabular@mode\z@}%
6418   {\chardef\bb@tabular@mode\@ne}% Mixed, with LTR cols
6419 \ifnum\bb@bidimode>\@ne % Any lua bidi= except default=1
6420 \ifcase\bb@tabular@mode\or % 1
6421   \let\bb@parabefore\relax
6422   \AddToHook{para/before}{\bb@parabefore}
6423   \AtBeginDocument{%
6424     \bb@replace\@tabular{$}{$%
6425       \def\bb@insidemath{0}%
6426       \def\bb@parabefore{\localerestoredirs}}%
6427     \ifnum\bb@tabular@mode=\@ne
6428       \bb@ifunset{@tabclassz}{}%
6429       \bb@exp{% Hide conditionals
6430         \\bb@sreplace\\@tabclassz
6431         {\<ifcase>\\@chnum}%
6432         {\localerestoredirs\<ifcase>\\@chnum}}}%
6433       \@ifpackageloaded{colortbl}%
6434       {\bb@sreplace\@classz
6435         {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6436       {\@ifpackageloaded{array}%
6437         {\bb@exp{% Hide conditionals
6438           \\bb@sreplace\\@classz
6439           {\<ifcase>\\@chnum}%
6440           {\bgroup\\localerestoredirs\<ifcase>\\@chnum}%
6441           \\bb@sreplace\\@classz
6442           {\do@row@strut\<fi>}{\do@row@strut\<fi>\egroup}}}%
6443         {}}%
6444     \fi}%
6445 \or % 2
6446   \let\bb@parabefore\relax
6447   \AddToHook{para/before}{\bb@parabefore}%
6448   \AtBeginDocument{%
6449     \@ifpackageloaded{colortbl}%
6450     {\bb@replace\@tabular{$}{$%
6451       \def\bb@insidemath{0}%
6452       \def\bb@parabefore{\localerestoredirs}}%
6453     \bb@sreplace\@classz
6454     {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6455     {}}%
6456 \fi

```

Very likely the `\output` routine must be patched in a quite general way to make sure the `\bodydir` is set to `\pagedir`. Note outside `\output` they can be different (and often are). For the moment, two *ad hoc* changes.

```

6457 \AtBeginDocument{%
6458   \@ifpackageloaded{multicol}%
6459   {\toks@\expandafter{\multi@column@out}%
6460    \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6461   {}%
6462   \@ifpackageloaded{paracol}%
6463   {\edef\pcol@output{%
6464     \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6465   {}}%
6466 \fi
6467 \ifx\bb@opt@layout\@nnil\endinput\fi % if no layout

```

OMEGA provided a companion to `\mathdir` (`\nextfake`) for those cases where we did not want it to be applied, so that the writing direction of the main text was left unchanged. `\bb@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`.

```

6468 \ifnum\bb@bidimode>\z@ % Any bidi=
6469 \def\bb@nextfake#1{% non-local changes, use always inside a group!

```

```

6470 \bbl@exp{%
6471 \def\\bbl@insidemath{0}%
6472 \mathdir\the\bodydir
6473 #1% Once entered in math, set boxes to restore values
6474 \<ifmode>%
6475 \everyvbox{%
6476 \the\everyvbox
6477 \bodydir\the\bodydir
6478 \mathdir\the\mathdir
6479 \everyhbox{\the\everyhbox}%
6480 \everyvbox{\the\everyvbox}}%
6481 \everyhbox{%
6482 \the\everyhbox
6483 \bodydir\the\bodydir
6484 \mathdir\the\mathdir
6485 \everyhbox{\the\everyhbox}%
6486 \everyvbox{\the\everyvbox}}%
6487 \<fi>}}%
6488 \def\@hangfrom#1{%
6489 \setbox\@tempboxa\hbox{#{#1}}%
6490 \hangindent\wd\@tempboxa
6491 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6492 \shapemode\@ne
6493 \fi
6494 \noindent\box\@tempboxa}
6495 \fi
6496 \IfBabelLayout{tabular}
6497 {\let\bbl@OL@tabular\@tabular
6498 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6499 \let\bbl@NL@tabular\@tabular
6500 \AtBeginDocument{%
6501 \ifx\bbl@NL@tabular\@tabular\else
6502 \bbl@exp{\\in@{\\bbl@nextfake}{\@tabular}}}%
6503 \ifin\else
6504 \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6505 \fi
6506 \let\bbl@NL@tabular\@tabular
6507 \fi}}
6508 {}
6509 \IfBabelLayout{lists}
6510 {\let\bbl@OL@list\list
6511 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6512 \let\bbl@NL@list\list
6513 \def\bbl@listparshape#1#2#3{%
6514 \parshape #1 #2 #3 %
6515 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6516 \shapemode\tw@
6517 \fi}}
6518 {}
6519 \IfBabelLayout{graphics}
6520 {\let\bbl@pictresetdir\relax
6521 \def\bbl@pictsetdir#1{%
6522 \ifcase\bbl@thetextdir
6523 \let\bbl@pictresetdir\relax
6524 \else
6525 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6526 \or\textdir TLT
6527 \else\bodydir TLT \textdir TLT
6528 \fi
6529 % \textdir required in pgf:
6530 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6531 \fi}%
6532 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%

```

```

6533 \directlua{
6534   Babel.get_picture_dir = true
6535   Babel.picture_has_bidi = 0
6536   %
6537   function Babel.picture_dir (head)
6538     if not Babel.get_picture_dir then return head end
6539     if Babel.hlist_has_bidi(head) then
6540       Babel.picture_has_bidi = 1
6541     end
6542     return head
6543   end
6544   luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6545     "Babel.picture_dir")
6546 }%
6547 \AtBeginDocument{%
6548   \def\LS@rot{%
6549     \setbox\@outputbox\vbox{%
6550       \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6551   \long\def\put(#1,#2)#3{%
6552     \@killglue
6553     % Try:
6554     \ifx\bbp@pictresetdir\relax
6555       \def\bbp@tempc{0}%
6556     \else
6557       \directlua{
6558         Babel.get_picture_dir = true
6559         Babel.picture_has_bidi = 0
6560       }%
6561       \setbox\z@\hb@xt@\z@{%
6562         \@defaultunitsset\@tempdimc{#1}\unitlength
6563         \kern\@tempdimc
6564         #3\hss}% TODO: #3 executed twice (below). That's bad.
6565       \edef\bbp@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6566     \fi
6567     % Do:
6568     \@defaultunitsset\@tempdimc{#2}\unitlength
6569     \raise\@tempdimc\hb@xt@\z@{%
6570       \@defaultunitsset\@tempdimc{#1}\unitlength
6571       \kern\@tempdimc
6572       {\ifnum\bbp@tempc>\z@\bbp@pictresetdir\fi#3}\hss}%
6573     \ignorespaces}%
6574   \MakeRobust\put}%
6575 \AtBeginDocument
6576 {\AddToHook{cmd/diagbox@pict/before}{\let\bbp@pictsetdir\@gobble}%
6577 \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6578   \AddToHook{env/pgfpicture/begin}{\bbp@pictsetdir\@ne}%
6579   \bbp@add\pgfinterruptpicture{\bbp@pictresetdir}%
6580   \bbp@add\pgfsys@beginpicture{\bbp@pictsetdir\z@}%
6581 \fi
6582 \ifx\tikzpicture\@undefined\else
6583   \AddToHook{env/tikzpicture/begin}{\bbp@pictsetdir\tw@}%
6584   \bbp@add\tikz@atbegin@node{\bbp@pictresetdir}%
6585   \bbp@sreplace\tikz{\begingroup}{\begingroup\bbp@pictsetdir\tw@}%
6586 \fi
6587 \ifx\tcolorbox\@undefined\else
6588   \def\tcb@drawing@env@begin{%
6589     \csname tcb@before@tcb@split@state\endcsname
6590     \bbp@pictsetdir\tw@
6591     \begin{\kv tcb@graphenv}%
6592     \tcb@bbdraw%
6593     \tcb@apply@graph@patches
6594   }%
6595   \def\tcb@drawing@env@end{%

```

```

6596      \end{\kv tcb@graphenv}%
6597      \bbl@pictresetdir
6598      \csname tcb@after@\tcb@split@state\endcsname
6599      }%
6600      \fi
6601    }}
6602  {}

```

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`.

```

6603 \IfBabelLayout{counters*}%
6604   {\bbl@add\bbl@opt@layout{.counters.}%
6605    \directlua{
6606      luatexbase.add_to_callback("process_output_buffer",
6607        Babel.discard_sublr , "Babel.discard_sublr") }%
6608   }}
6609 \IfBabelLayout{counters}%
6610   {\let\bbl@0L@@textsuperscript\@textsuperscript
6611    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6612    \let\bbl@latinarabic=\@arabic
6613    \let\bbl@0L@@arabic\@arabic
6614    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6615    \@ifpackagewith{babel}{bidi=default}%
6616      {\let\bbl@asciroman=\@roman
6617       \let\bbl@0L@@roman\@roman
6618       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6619       \let\bbl@asciiRoman=\@Roman
6620       \let\bbl@0L@@roman\@Roman
6621       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6622       \let\bbl@0L@labelenumii\labelenumii
6623       \def\labelenumii{}\theenumii}%
6624       \let\bbl@0L@p@enumiii\p@enumiii
6625       \def\p@enumiii{\p@enumii}\theenumii{}\}}{}
6626 <<Footnote changes>>
6627 \IfBabelLayout{footnotes}%
6628   {\let\bbl@0L@footnote\footnote
6629    \BabelFootnote\footnote\language\language{}{}%
6630    \BabelFootnote\localfootnote\language\language{}{}%
6631    \BabelFootnote\mainfootnote{}{}{}%
6632   {}

```

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

```

6633 \IfBabelLayout{extras}%
6634   {\bbl@ncarg\let\bbl@0L@underline\underline }%
6635   \bbl@carg\bbl@sreplace{underline }%
6636     {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
6637   \bbl@carg\bbl@sreplace{underline }%
6638     {\m@th$}{\m@th$\egroup}%
6639   \let\bbl@0L@LaTeXe\LaTeXe
6640   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6641     \if b\expandafter\@car\@series\@nil\boldmath\fi
6642     \babelsublr{%
6643       \LaTeXe\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}%
6644   {}
6645 </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.

```

6646 (*transforms)
6647 Babel.linebreaking.replacements = {}
6648 Babel.linebreaking.replacements[0] = {} -- pre
6649 Babel.linebreaking.replacements[1] = {} -- post
6650
6651 -- Discretionaries contain strings as nodes
6652 function Babel.str_to_nodes(fn, matches, base)
6653   local n, head, last
6654   if fn == nil then return nil end
6655   for s in string.utfvalues(fn(matches)) do
6656     if base.id == 7 then
6657       base = base.replace
6658     end
6659     n = node.copy(base)
6660     n.char = s
6661     if not head then
6662       head = n
6663     else
6664       last.next = n
6665     end
6666     last = n
6667   end
6668   return head
6669 end
6670
6671 Babel.fetch_subtext = {}
6672
6673 Babel.ignore_pre_char = function(node)
6674   return (node.lang == Babel.nohyphenation)
6675 end
6676
6677 -- Merging both functions doesn't seem feasible, because there are too
6678 -- many differences.
6679 Babel.fetch_subtext[0] = function(head)
6680   local word_string = ''
6681   local word_nodes = {}
6682   local lang
6683   local item = head
6684   local inmath = false
6685
6686   while item do
6687     if item.id == 11 then
6688       inmath = (item.subtype == 0)
6689     end
6690
6691     if inmath then
6692       -- pass
6693     end
6694
6695     elseif item.id == 29 then
6696       local locale = node.get_attribute(item, Babel.attr_locale)
6697
6698       if lang == locale or lang == nil then
6699         lang = lang or locale
6700         if Babel.ignore_pre_char(item) then

```



```

6701         word_string = word_string .. Babel.us_char
6702     else
6703         word_string = word_string .. unicode.utf8.char(item.char)
6704     end
6705     word_nodes[#word_nodes+1] = item
6706     else
6707         break
6708     end
6709
6710     elseif item.id == 12 and item.subtype == 13 then
6711         word_string = word_string .. ' '
6712         word_nodes[#word_nodes+1] = item
6713
6714     -- Ignore leading unrecognized nodes, too.
6715     elseif word_string ~= '' then
6716         word_string = word_string .. Babel.us_char
6717         word_nodes[#word_nodes+1] = item -- Will be ignored
6718     end
6719
6720     item = item.next
6721 end
6722
6723 -- Here and above we remove some trailing chars but not the
6724 -- corresponding nodes. But they aren't accessed.
6725 if word_string:sub(-1) == ' ' then
6726     word_string = word_string:sub(1,-2)
6727 end
6728 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6729 return word_string, word_nodes, item, lang
6730 end
6731
6732 Babel.fetch_subtext[1] = function(head)
6733     local word_string = ''
6734     local word_nodes = {}
6735     local lang
6736     local item = head
6737     local inmath = false
6738
6739     while item do
6740
6741         if item.id == 11 then
6742             inmath = (item.subtype == 0)
6743         end
6744
6745         if inmath then
6746             -- pass
6747
6748         elseif item.id == 29 then
6749             if item.lang == lang or lang == nil then
6750                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6751                     lang = lang or item.lang
6752                     word_string = word_string .. unicode.utf8.char(item.char)
6753                     word_nodes[#word_nodes+1] = item
6754                 end
6755             else
6756                 break
6757             end
6758
6759         elseif item.id == 7 and item.subtype == 2 then
6760             word_string = word_string .. '='
6761             word_nodes[#word_nodes+1] = item
6762
6763         elseif item.id == 7 and item.subtype == 3 then

```

```

6764     word_string = word_string .. '|'
6765     word_nodes[#word_nodes+1] = item
6766
6767     -- (1) Go to next word if nothing was found, and (2) implicitly
6768     -- remove leading USs.
6769     elseif word_string == '' then
6770         -- pass
6771
6772     -- This is the responsible for splitting by words.
6773     elseif (item.id == 12 and item.subtype == 13) then
6774         break
6775
6776     else
6777         word_string = word_string .. Babel.us_char
6778         word_nodes[#word_nodes+1] = item -- Will be ignored
6779     end
6780
6781     item = item.next
6782 end
6783
6784 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6785 return word_string, word_nodes, item, lang
6786 end
6787
6788 function Babel.pre_hyphenate_replace(head)
6789     Babel.hyphenate_replace(head, 0)
6790 end
6791
6792 function Babel.post_hyphenate_replace(head)
6793     Babel.hyphenate_replace(head, 1)
6794 end
6795
6796 Babel.us_char = string.char(31)
6797
6798 function Babel.hyphenate_replace(head, mode)
6799     local u = unicode.utf8
6800     local lbkr = Babel.linebreaking.replacements[mode]
6801
6802     local word_head = head
6803
6804     while true do -- for each subtext block
6805
6806         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6807
6808         if Babel.debug then
6809             print()
6810             print((mode == 0) and '@@@<' or '@@@>', w)
6811         end
6812
6813         if nw == nil and w == '' then break end
6814
6815         if not lang then goto next end
6816         if not lbkr[lang] then goto next end
6817
6818         -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6819         -- loops are nested.
6820         for k=1, #lbkr[lang] do
6821             local p = lbkr[lang][k].pattern
6822             local r = lbkr[lang][k].replace
6823             local attr = lbkr[lang][k].attr or -1
6824
6825             if Babel.debug then
6826                 print('*****', p, mode)

```

```

6827     end
6828
6829     -- This variable is set in some cases below to the first *byte*
6830     -- after the match, either as found by u.match (faster) or the
6831     -- computed position based on sc if w has changed.
6832     local last_match = 0
6833     local step = 0
6834
6835     -- For every match.
6836     while true do
6837         if Babel.debug then
6838             print('====')
6839         end
6840         local new -- used when inserting and removing nodes
6841
6842         local matches = { u.match(w, p, last_match) }
6843
6844         if #matches < 2 then break end
6845
6846         -- Get and remove empty captures (with ())'s, which return a
6847         -- number with the position), and keep actual captures
6848         -- (from (...)), if any, in matches.
6849         local first = table.remove(matches, 1)
6850         local last = table.remove(matches, #matches)
6851         -- Non re-fetched substrings may contain \31, which separates
6852         -- subsubstrings.
6853         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6854
6855         local save_last = last -- with A()BC()D, points to D
6856
6857         -- Fix offsets, from bytes to unicode. Explained above.
6858         first = u.len(w:sub(1, first-1)) + 1
6859         last = u.len(w:sub(1, last-1)) -- now last points to C
6860
6861         -- This loop stores in a small table the nodes
6862         -- corresponding to the pattern. Used by 'data' to provide a
6863         -- predictable behavior with 'insert' (w_nodes is modified on
6864         -- the fly), and also access to 'remove'd nodes.
6865         local sc = first-1 -- Used below, too
6866         local data_nodes = {}
6867
6868         local enabled = true
6869         for q = 1, last-first+1 do
6870             data_nodes[q] = w_nodes[sc+q]
6871             if enabled
6872                 and attr > -1
6873                 and not node.has_attribute(data_nodes[q], attr)
6874             then
6875                 enabled = false
6876             end
6877         end
6878
6879         -- This loop traverses the matched substring and takes the
6880         -- corresponding action stored in the replacement list.
6881         -- sc = the position in substr nodes / string
6882         -- rc = the replacement table index
6883         local rc = 0
6884
6885         while rc < last-first+1 do -- for each replacement
6886             if Babel.debug then
6887                 print('.....', rc + 1)
6888             end
6889             sc = sc + 1

```

```

6890         rc = rc + 1
6891
6892     if Babel.debug then
6893         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6894         local ss = ''
6895         for itt in node.traverse(head) do
6896             if itt.id == 29 then
6897                 ss = ss .. unicode.utf8.char(itt.char)
6898             else
6899                 ss = ss .. '{' .. itt.id .. '}'
6900             end
6901         end
6902         print('*****', ss)
6903
6904     end
6905
6906     local crep = r[rc]
6907     local item = w_nodes[sc]
6908     local item_base = item
6909     local placeholder = Babel.us_char
6910     local d
6911
6912     if crep and crep.data then
6913         item_base = data_nodes[crep.data]
6914     end
6915
6916     if crep then
6917         step = crep.step or 0
6918     end
6919
6920     if (not enabled) or (crep and next(crep) == nil) then -- = {}
6921         last_match = save_last -- Optimization
6922         goto next
6923
6924     elseif crep == nil or crep.remove then
6925         node.remove(head, item)
6926         table.remove(w_nodes, sc)
6927         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6928         sc = sc - 1 -- Nothing has been inserted.
6929         last_match = utf8.offset(w, sc+1+step)
6930         goto next
6931
6932     elseif crep and crep.kashida then -- Experimental
6933         node.set_attribute(item,
6934             Babel.attr_kashida,
6935             crep.kashida)
6936         last_match = utf8.offset(w, sc+1+step)
6937         goto next
6938
6939     elseif crep and crep.string then
6940         local str = crep.string(matches)
6941         if str == '' then -- Gather with nil
6942             node.remove(head, item)
6943             table.remove(w_nodes, sc)
6944             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6945             sc = sc - 1 -- Nothing has been inserted.
6946         else
6947             local loop_first = true
6948             for s in string.utfvalues(str) do
6949                 d = node.copy(item_base)
6950                 d.char = s
6951                 if loop_first then
6952                     loop_first = false

```

```

6953         head, new = node.insert_before(head, item, d)
6954         if sc == 1 then
6955             word_head = head
6956         end
6957         w_nodes[sc] = d
6958         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6959     else
6960         sc = sc + 1
6961         head, new = node.insert_before(head, item, d)
6962         table.insert(w_nodes, sc, new)
6963         w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6964     end
6965     if Babel.debug then
6966         print('.....', 'str')
6967         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6968     end
6969     end -- for
6970     node.remove(head, item)
6971 end -- if ''
6972 last_match = utf8.offset(w, sc+1+step)
6973 goto next
6974
6975 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6976     d = node.new(7, 3) -- (disc, regular)
6977     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6978     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6979     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6980     d.attr = item_base.attr
6981     if crep.pre == nil then -- TeXbook p96
6982         d.penalty = crep.penalty or tex.hyphenpenalty
6983     else
6984         d.penalty = crep.penalty or tex.exhyphenpenalty
6985     end
6986     placeholder = '|'
6987     head, new = node.insert_before(head, item, d)
6988
6989 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6990     -- ERROR
6991
6992 elseif crep and crep.penalty then
6993     d = node.new(14, 0) -- (penalty, userpenalty)
6994     d.attr = item_base.attr
6995     d.penalty = crep.penalty
6996     head, new = node.insert_before(head, item, d)
6997
6998 elseif crep and crep.space then
6999     -- 655360 = 10 pt = 10 * 65536 sp
7000     d = node.new(12, 13) -- (glue, spaceskip)
7001     local quad = font.getfont(item_base.font).size or 655360
7002     node.setglue(d, crep.space[1] * quad,
7003                 crep.space[2] * quad,
7004                 crep.space[3] * quad)
7005     if mode == 0 then
7006         placeholder = ' '
7007     end
7008     head, new = node.insert_before(head, item, d)
7009
7010 elseif crep and crep.spacefactor then
7011     d = node.new(12, 13) -- (glue, spaceskip)
7012     local base_font = font.getfont(item_base.font)
7013     node.setglue(d,
7014                 crep.spacefactor[1] * base_font.parameters['space'],
7015                 crep.spacefactor[2] * base_font.parameters['space_stretch'],

```

```

7016         crep.spacefactor[3] * base_font.parameters['space_shrink'])
7017     if mode == 0 then
7018         placeholder = ' '
7019     end
7020     head, new = node.insert_before(head, item, d)
7021
7022     elseif mode == 0 and crep and crep.space then
7023         -- ERROR
7024
7025     end -- ie replacement cases
7026
7027     -- Shared by disc, space and penalty.
7028     if sc == 1 then
7029         word_head = head
7030     end
7031     if crep.insert then
7032         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7033         table.insert(w_nodes, sc, new)
7034         last = last + 1
7035     else
7036         w_nodes[sc] = d
7037         node.remove(head, item)
7038         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7039     end
7040
7041     last_match = utf8.offset(w, sc+1+step)
7042
7043     ::next::
7044
7045     end -- for each replacement
7046
7047     if Babel.debug then
7048         print('.....', '/')
7049         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7050     end
7051
7052     end -- for match
7053
7054     end -- for patterns
7055
7056     ::next::
7057     word_head = nw
7058 end -- for substring
7059 return head
7060 end
7061
7062 -- This table stores capture maps, numbered consecutively
7063 Babel.capture_maps = {}
7064
7065 -- The following functions belong to the next macro
7066 function Babel.capture_func(key, cap)
7067     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..[") .. "]"
7068     local cnt
7069     local u = unicode.utf8
7070     ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
7071     if cnt == 0 then
7072         ret = u.gsub(ret, '{(%x%x%x%x+)}',
7073             function (n)
7074                 return u.char(tonumber(n, 16))
7075             end)
7076     end
7077     ret = ret:gsub("%[%[%]]%.%", '')
7078     ret = ret:gsub("%.%.%[%[%]]%", '')

```

```

7079 return key .. [{"=function(m) return }} .. ret .. [{" end}}
7080 end
7081
7082 function Babel.capt_map(from, mapno)
7083 return Babel.capture_maps[mapno][from] or from
7084 end
7085
7086 -- Handle the {n|abc|ABC} syntax in captures
7087 function Babel.capture_func_map(capno, from, to)
7088 local u = unicode.utf8
7089 from = u.gsub(from, '{(%x%x%x%x+)}',
7090 function (n)
7091 return u.char(tonumber(n, 16))
7092 end)
7093 to = u.gsub(to, '{(%x%x%x%x+)}',
7094 function (n)
7095 return u.char(tonumber(n, 16))
7096 end)
7097 local froms = {}
7098 for s in string.utfcharacters(from) do
7099 table.insert(froms, s)
7100 end
7101 local cnt = 1
7102 table.insert(Babel.capture_maps, {})
7103 local mlen = table.getn(Babel.capture_maps)
7104 for s in string.utfcharacters(to) do
7105 Babel.capture_maps[mlen][froms[cnt]] = s
7106 cnt = cnt + 1
7107 end
7108 return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7109 (mlen) .. ").." .. "[["
7110 end
7111
7112 -- Create/Extend reversed sorted list of kashida weights:
7113 function Babel.capture_kashida(key, wt)
7114 wt = tonumber(wt)
7115 if Babel.kashida_wts then
7116 for p, q in ipairs(Babel.kashida_wts) do
7117 if wt == q then
7118 break
7119 elseif wt > q then
7120 table.insert(Babel.kashida_wts, p, wt)
7121 break
7122 elseif table.getn(Babel.kashida_wts) == p then
7123 table.insert(Babel.kashida_wts, wt)
7124 end
7125 end
7126 else
7127 Babel.kashida_wts = { wt }
7128 end
7129 return 'kashida = ' .. wt
7130 end
7131
7132 -- Experimental: applies prehyphenation transforms to a string (letters
7133 -- and spaces).
7134 function Babel.string_prehyphenation(str, locale)
7135 local n, head, last, res
7136 head = node.new(8, 0) -- dummy (hack just to start)
7137 last = head
7138 for s in string.utfvalues(str) do
7139 if s == 20 then
7140 n = node.new(12, 0)
7141 else

```

```

7142     n = node.new(29, 0)
7143     n.char = s
7144     end
7145     node.set_attribute(n, Babel.attr_locale, locale)
7146     last.next = n
7147     last = n
7148   end
7149   head = Babel.hyphenate_replace(head, 0)
7150   res = ''
7151   for n in node.traverse(head) do
7152     if n.id == 12 then
7153       res = res .. ' '
7154     elseif n.id == 29 then
7155       res = res .. unicode.utf8.char(n.char)
7156     end
7157   end
7158   tex.print(res)
7159 end
7160 </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.

```

7161 (*basic-r)
7162 Babel = Babel or {}
7163
7164 Babel.bidi_enabled = true
7165

```



```

7166 require('babel-data-bidi.lua')
7167
7168 local characters = Babel.characters
7169 local ranges = Babel.ranges
7170
7171 local DIR = node.id("dir")
7172
7173 local function dir_mark(head, from, to, outer)
7174   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7175   local d = node.new(DIR)
7176   d.dir = '+' .. dir
7177   node.insert_before(head, from, d)
7178   d = node.new(DIR)
7179   d.dir = '-' .. dir
7180   node.insert_after(head, to, d)
7181 end
7182
7183 function Babel.bidi(head, ispar)
7184   local first_n, last_n          -- first and last char with nums
7185   local last_es                  -- an auxiliary 'last' used with nums
7186   local first_d, last_d          -- first and last char in L/R block
7187   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):

```

7188   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7189   local strong_lr = (strong == 'l') and 'l' or 'r'
7190   local outer = strong
7191
7192   local new_dir = false
7193   local first_dir = false
7194   local inmath = false
7195
7196   local last_lr
7197
7198   local type_n = ''
7199
7200   for item in node.traverse(head) do
7201
7202     -- three cases: glyph, dir, otherwise
7203     if item.id == node.id'glyph'
7204       or (item.id == 7 and item.subtype == 2) then
7205
7206       local itemchar
7207       if item.id == 7 and item.subtype == 2 then
7208         itemchar = item.replace.char
7209       else
7210         itemchar = item.char
7211       end
7212       local chardata = characters[itemchar]
7213       dir = chardata and chardata.d or nil
7214       if not dir then
7215         for nn, et in ipairs(ranges) do
7216           if itemchar < et[1] then
7217             break
7218           elseif itemchar <= et[2] then
7219             dir = et[3]
7220             break
7221           end
7222         end
7223       end
7224       dir = dir or 'l'

```

```
7225     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).

```
7226     if new_dir then
7227         attr_dir = 0
7228         for at in node.traverse(item.attr) do
7229             if at.number == Babel.attr_dir then
7230                 attr_dir = at.value & 0x3
7231             end
7232         end
7233         if attr_dir == 1 then
7234             strong = 'r'
7235         elseif attr_dir == 2 then
7236             strong = 'al'
7237         else
7238             strong = 'l'
7239         end
7240         strong_lr = (strong == 'l') and 'l' or 'r'
7241         outer = strong_lr
7242         new_dir = false
7243     end
7244
7245     if dir == 'nsm' then dir = strong end -- W1
```

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

```
7246     dir_real = dir -- We need dir_real to set strong below
7247     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:

```
7248     if strong == 'al' then
7249         if dir == 'en' then dir = 'an' end -- W2
7250         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7251         strong_lr = 'r' -- W3
7252     end
```

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

```
7253     elseif item.id == node.id'dir' and not inmath then
7254         new_dir = true
7255         dir = nil
7256     elseif item.id == node.id'math' then
7257         inmath = (item.subtype == 0)
7258     else
7259         dir = nil -- Not a char
7260     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>.

```
7261     if dir == 'en' or dir == 'an' or dir == 'et' then
7262         if dir ~= 'et' then
7263             type_n = dir
7264         end
7265         first_n = first_n or item
7266         last_n = last_es or item
7267         last_es = nil
7268     elseif dir == 'es' and last_n then -- W3+W6
7269         last_es = item
```

```

7270 elseif dir == 'cs' then          -- it's right - do nothing
7271 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7272   if strong_lr == 'r' and type_n ~= '' then
7273     dir_mark(head, first_n, last_n, 'r')
7274   elseif strong_lr == 'l' and first_d and type_n == 'an' then
7275     dir_mark(head, first_n, last_n, 'r')
7276     dir_mark(head, first_d, last_d, outer)
7277     first_d, last_d = nil, nil
7278   elseif strong_lr == 'l' and type_n ~= '' then
7279     last_d = last_n
7280   end
7281   type_n = ''
7282   first_n, last_n = nil, nil
7283 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:

```

7284   if dir == 'l' or dir == 'r' then
7285     if dir ~= outer then
7286       first_d = first_d or item
7287       last_d = item
7288     elseif first_d and dir ~= strong_lr then
7289       dir_mark(head, first_d, last_d, outer)
7290       first_d, last_d = nil, nil
7291     end
7292   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.

```

7293   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7294     item.char = characters[item.char] and
7295       characters[item.char].m or item.char
7296   elseif (dir or new_dir) and last_lr ~= item then
7297     local mir = outer .. strong_lr .. (dir or outer)
7298     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7299       for ch in node.traverse(node.next(last_lr)) do
7300         if ch == item then break end
7301         if ch.id == node.id'glyph' and characters[ch.char] then
7302           ch.char = characters[ch.char].m or ch.char
7303         end
7304       end
7305     end
7306   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).

```

7307   if dir == 'l' or dir == 'r' then
7308     last_lr = item
7309     strong = dir_real          -- Don't search back - best save now
7310     strong_lr = (strong == 'l') and 'l' or 'r'
7311   elseif new_dir then
7312     last_lr = nil
7313   end
7314 end

```

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

```

7315   if last_lr and outer == 'r' then
7316     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7317       if characters[ch.char] then

```

```

7318         ch.char = characters[ch.char].m or ch.char
7319     end
7320 end
7321 end
7322 if first_n then
7323     dir_mark(head, first_n, last_n, outer)
7324 end
7325 if first_d then
7326     dir_mark(head, first_d, last_d, outer)
7327 end

```

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

```

7328 return node.prev(head) or head
7329 end
7330 </basic-r>

```

And here the Lua code for bidi=basic:

```

7331 <(*basic)
7332 Babel = Babel or {}
7333
7334 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7335
7336 Babel.fontmap = Babel.fontmap or {}
7337 Babel.fontmap[0] = {}      -- l
7338 Babel.fontmap[1] = {}      -- r
7339 Babel.fontmap[2] = {}      -- al/an
7340
7341 Babel.bidi_enabled = true
7342 Babel.mirroring_enabled = true
7343
7344 require('babel-data-bidi.lua')
7345
7346 local characters = Babel.characters
7347 local ranges = Babel.ranges
7348
7349 local DIR = node.id('dir')
7350 local GLYPH = node.id('glyph')
7351
7352 local function insert_implicit(head, state, outer)
7353     local new_state = state
7354     if state.sim and state.eim and state.sim ~= state.eim then
7355         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7356         local d = node.new(DIR)
7357         d.dir = '+' .. dir
7358         node.insert_before(head, state.sim, d)
7359         local d = node.new(DIR)
7360         d.dir = '-' .. dir
7361         node.insert_after(head, state.eim, d)
7362     end
7363     new_state.sim, new_state.eim = nil, nil
7364     return head, new_state
7365 end
7366
7367 local function insert_numeric(head, state)
7368     local new
7369     local new_state = state
7370     if state.san and state.ean and state.san ~= state.ean then
7371         local d = node.new(DIR)
7372         d.dir = '+TLT'
7373         _, new = node.insert_before(head, state.san, d)
7374         if state.san == state.sim then state.sim = new end
7375         local d = node.new(DIR)
7376         d.dir = '-TLT'

```

```

7377     _, new = node.insert_after(head, state.ean, d)
7378     if state.ean == state.eim then state.eim = new end
7379 end
7380 new_state.san, new_state.ean = nil, nil
7381 return head, new_state
7382 end
7383
7384 -- TODO - \hbox with an explicit dir can lead to wrong results
7385 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7386 -- was s made to improve the situation, but the problem is the 3-dir
7387 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7388 -- well.
7389
7390 function Babel.bidi(head, ispar, hdir)
7391     local d -- d is used mainly for computations in a loop
7392     local prev_d = ''
7393     local new_d = false
7394
7395     local nodes = {}
7396     local outer_first = nil
7397     local inmath = false
7398
7399     local glue_d = nil
7400     local glue_i = nil
7401
7402     local has_en = false
7403     local first_et = nil
7404
7405     local has_hyperlink = false
7406
7407     local ATDIR = Babel.attr_dir
7408
7409     local save_outer
7410     local temp = node.get_attribute(head, ATDIR)
7411     if temp then
7412         temp = temp & 0x3
7413         save_outer = (temp == 0 and 'l') or
7414                     (temp == 1 and 'r') or
7415                     (temp == 2 and 'al')
7416     elseif ispar then -- Or error? Shouldn't happen
7417         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7418     else -- Or error? Shouldn't happen
7419         save_outer = ('TRT' == hdir) and 'r' or 'l'
7420     end
7421     -- when the callback is called, we are just _after_ the box,
7422     -- and the textdir is that of the surrounding text
7423     -- if not ispar and hdir ~= tex.textdir then
7424     --     save_outer = ('TRT' == hdir) and 'r' or 'l'
7425     -- end
7426     local outer = save_outer
7427     local last = outer
7428     -- 'al' is only taken into account in the first, current loop
7429     if save_outer == 'al' then save_outer = 'r' end
7430
7431     local fontmap = Babel.fontmap
7432
7433     for item in node.traverse(head) do
7434
7435         -- In what follows, #node is the last (previous) node, because the
7436         -- current one is not added until we start processing the neutrals.
7437
7438         -- three cases: glyph, dir, otherwise
7439         if item.id == GLYPH

```

```

7440         or (item.id == 7 and item.subtype == 2) then
7441
7442         local d_font = nil
7443         local item_r
7444         if item.id == 7 and item.subtype == 2 then
7445             item_r = item.replace    -- automatic discs have just 1 glyph
7446         else
7447             item_r = item
7448         end
7449         local chardata = characters[item_r.char]
7450         d = chardata and chardata.d or nil
7451         if not d or d == 'nsm' then
7452             for nn, et in ipairs(ranges) do
7453                 if item_r.char < et[1] then
7454                     break
7455                 elseif item_r.char <= et[2] then
7456                     if not d then d = et[3]
7457                     elseif d == 'nsm' then d_font = et[3]
7458                     end
7459                     break
7460                 end
7461             end
7462         end
7463         d = d or 'l'
7464
7465         -- A short 'pause' in bidi for mapfont
7466         d_font = d_font or d
7467         d_font = (d_font == 'l' and 0) or
7468                 (d_font == 'nsm' and 0) or
7469                 (d_font == 'r' and 1) or
7470                 (d_font == 'al' and 2) or
7471                 (d_font == 'an' and 2) or nil
7472         if d_font and fontmap and fontmap[d_font][item_r.font] then
7473             item_r.font = fontmap[d_font][item_r.font]
7474         end
7475
7476         if new_d then
7477             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7478             if inmath then
7479                 attr_d = 0
7480             else
7481                 attr_d = node.get_attribute(item, ATDIR)
7482                 attr_d = attr_d & 0x3
7483             end
7484             if attr_d == 1 then
7485                 outer_first = 'r'
7486                 last = 'r'
7487             elseif attr_d == 2 then
7488                 outer_first = 'r'
7489                 last = 'al'
7490             else
7491                 outer_first = 'l'
7492                 last = 'l'
7493             end
7494             outer = last
7495             has_en = false
7496             first_et = nil
7497             new_d = false
7498         end
7499
7500         if glue_d then
7501             if (d == 'l' and 'l' or 'r') ~= glue_d then
7502                 table.insert(nodes, {glue_i, 'on', nil})

```

```

7503         end
7504         glue_d = nil
7505         glue_i = nil
7506     end
7507
7508     elseif item.id == DIR then
7509         d = nil
7510
7511         if head ~= item then new_d = true end
7512
7513     elseif item.id == node.id'glue' and item.subtype == 13 then
7514         glue_d = d
7515         glue_i = item
7516         d = nil
7517
7518     elseif item.id == node.id'math' then
7519         inmath = (item.subtype == 0)
7520
7521     elseif item.id == 8 and item.subtype == 19 then
7522         has_hyperlink = true
7523
7524     else
7525         d = nil
7526     end
7527
7528     -- AL <= EN/ET/ES      -- W2 + W3 + W6
7529     if last == 'al' and d == 'en' then
7530         d = 'an'          -- W3
7531     elseif last == 'al' and (d == 'et' or d == 'es') then
7532         d = 'on'          -- W6
7533     end
7534
7535     -- EN + CS/ES + EN      -- W4
7536     if d == 'en' and #nodes >= 2 then
7537         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7538             and nodes[#nodes-1][2] == 'en' then
7539             nodes[#nodes][2] = 'en'
7540         end
7541     end
7542
7543     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7544     if d == 'an' and #nodes >= 2 then
7545         if (nodes[#nodes][2] == 'cs')
7546             and nodes[#nodes-1][2] == 'an' then
7547             nodes[#nodes][2] = 'an'
7548         end
7549     end
7550
7551     -- ET/EN                  -- W5 + W7->l / W6->on
7552     if d == 'et' then
7553         first_et = first_et or (#nodes + 1)
7554     elseif d == 'en' then
7555         has_en = true
7556         first_et = first_et or (#nodes + 1)
7557     elseif first_et then      -- d may be nil here !
7558         if has_en then
7559             if last == 'l' then
7560                 temp = 'l'    -- W7
7561             else
7562                 temp = 'en'   -- W5
7563             end
7564         else
7565             temp = 'on'       -- W6

```

```

7566     end
7567     for e = first_et, #nodes do
7568         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7569     end
7570     first_et = nil
7571     has_en = false
7572 end
7573
7574 -- Force mathdir in math if ON (currently works as expected only
7575 -- with 'l')
7576 if inmath and d == 'on' then
7577     d = ('TRT' == tex.mathdir) and 'r' or 'l'
7578 end
7579
7580 if d then
7581     if d == 'al' then
7582         d = 'r'
7583         last = 'al'
7584     elseif d == 'l' or d == 'r' then
7585         last = d
7586     end
7587     prev_d = d
7588     table.insert(nodes, {item, d, outer_first})
7589 end
7590
7591 outer_first = nil
7592
7593 end
7594
7595 -- TODO -- repeated here in case EN/ET is the last node. Find a
7596 -- better way of doing things:
7597 if first_et then -- dir may be nil here !
7598     if has_en then
7599         if last == 'l' then
7600             temp = 'l' -- W7
7601         else
7602             temp = 'en' -- W5
7603         end
7604     else
7605         temp = 'on' -- W6
7606     end
7607     for e = first_et, #nodes do
7608         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7609     end
7610 end
7611
7612 -- dummy node, to close things
7613 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7614
7615 ----- NEUTRAL -----
7616
7617 outer = save_outer
7618 last = outer
7619
7620 local first_on = nil
7621
7622 for q = 1, #nodes do
7623     local item
7624
7625     local outer_first = nodes[q][3]
7626     outer = outer_first or outer
7627     last = outer_first or last
7628

```



```

7629     local d = nodes[q][2]
7630     if d == 'an' or d == 'en' then d = 'r' end
7631     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7632
7633     if d == 'on' then
7634         first_on = first_on or q
7635     elseif first_on then
7636         if last == d then
7637             temp = d
7638         else
7639             temp = outer
7640         end
7641         for r = first_on, q - 1 do
7642             nodes[r][2] = temp
7643             item = nodes[r][1] -- MIRRORING
7644             if Babel.mirroring_enabled and item.id == GLYPH
7645                 and temp == 'r' and characters[item.char] then
7646                 local font_mode = ''
7647                 if item.font > 0 and font.fonts[item.font].properties then
7648                     font_mode = font.fonts[item.font].properties.mode
7649                 end
7650                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7651                     item.char = characters[item.char].m or item.char
7652                 end
7653             end
7654         end
7655         first_on = nil
7656     end
7657
7658     if d == 'r' or d == 'l' then last = d end
7659 end
7660
7661 ----- IMPLICIT, REORDER -----
7662
7663 outer = save_outer
7664 last = outer
7665
7666 local state = {}
7667 state.has_r = false
7668
7669 for q = 1, #nodes do
7670
7671     local item = nodes[q][1]
7672
7673     outer = nodes[q][3] or outer
7674
7675     local d = nodes[q][2]
7676
7677     if d == 'nsm' then d = last end -- W1
7678     if d == 'en' then d = 'an' end
7679     local isdir = (d == 'r' or d == 'l')
7680
7681     if outer == 'l' and d == 'an' then
7682         state.san = state.san or item
7683         state.ean = item
7684     elseif state.san then
7685         head, state = insert_numeric(head, state)
7686     end
7687
7688     if outer == 'l' then
7689         if d == 'an' or d == 'r' then -- im -> implicit
7690             if d == 'r' then state.has_r = true end
7691             state.sim = state.sim or item

```

```

7692         state.eim = item
7693     elseif d == 'l' and state.sim and state.has_r then
7694         head, state = insert_implicit(head, state, outer)
7695     elseif d == 'l' then
7696         state.sim, state.eim, state.has_r = nil, nil, false
7697     end
7698 else
7699     if d == 'an' or d == 'l' then
7700         if nodes[q][3] then -- nil except after an explicit dir
7701             state.sim = item -- so we move sim 'inside' the group
7702         else
7703             state.sim = state.sim or item
7704         end
7705         state.eim = item
7706     elseif d == 'r' and state.sim then
7707         head, state = insert_implicit(head, state, outer)
7708     elseif d == 'r' then
7709         state.sim, state.eim = nil, nil
7710     end
7711 end
7712
7713 if isdir then
7714     last = d -- Don't search back - best save now
7715 elseif d == 'on' and state.san then
7716     state.san = state.san or item
7717     state.ean = item
7718 end
7719
7720 end
7721
7722 head = node.prev(head) or head
7723
7724 ----- FIX HYPERLINKS -----
7725
7726 if has_hyperlink then
7727     local flag, linking = 0, 0
7728     for item in node.traverse(head) do
7729         if item.id == DIR then
7730             if item.dir == '+TRT' or item.dir == '+TLT' then
7731                 flag = flag + 1
7732             elseif item.dir == '-TRT' or item.dir == '-TLT' then
7733                 flag = flag - 1
7734             end
7735         elseif item.id == 8 and item.subtype == 19 then
7736             linking = flag
7737         elseif item.id == 8 and item.subtype == 20 then
7738             if linking > 0 then
7739                 if item.prev.id == DIR and
7740                     (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7741                     d = node.new(DIR)
7742                     d.dir = item.prev.dir
7743                     node.remove(head, item.prev)
7744                     node.insert_after(head, item, d)
7745                 end
7746             end
7747             linking = 0
7748         end
7749     end
7750 end
7751
7752 return head
7753 end
7754 </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.

```
7755 ⟨*nil⟩
7756 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ v⟨⟨version⟩⟩ Nil language]
7757 \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.

```
7758 \ifx\l@nil\@undefined
7759   \newlanguage\l@nil
7760   \@namedef{bbl@hyphendata@the\l@nil}{}{}% Remove warning
7761   \let\bbl@elt\relax
7762   \edef\bbl@languages{% Add it to the list of languages
7763     \bbl@languages\bbl@elt{nil}{the\l@nil}{}{}}
7764 \fi
```

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

```
7765 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7766 \let\captionnil\@empty
7767 \let\datenil\@empty
```

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

```
7768 \def\bbl@inidata@nil{%
7769   \bbl@elt{identification}{tag.ini}{und}%
7770   \bbl@elt{identification}{load.level}{0}%
7771   \bbl@elt{identification}{charset}{utf8}%
7772   \bbl@elt{identification}{version}{1.0}%
7773   \bbl@elt{identification}{date}{2022-05-16}%
7774   \bbl@elt{identification}{name.local}{nil}%
7775   \bbl@elt{identification}{name.english}{nil}%
7776   \bbl@elt{identification}{name.babel}{nil}%
7777   \bbl@elt{identification}{tag.bcp47}{und}%
7778   \bbl@elt{identification}{language.tag.bcp47}{und}%
7779   \bbl@elt{identification}{tag.opentype}{dflt}%
7780   \bbl@elt{identification}{script.name}{Latin}%
7781   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7782   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7783   \bbl@elt{identification}{level}{1}%
7784   \bbl@elt{identification}{encodings}{}%
7785   \bbl@elt{identification}{derive}{}{no}}
```

```

7786 \@namedef{bbl@tbc@nil}{und}
7787 \@namedef{bbl@lbcp@nil}{und}
7788 \@namedef{bbl@cas@nil}{und} % TODO
7789 \@namedef{bbl@lotf@nil}{dflt}
7790 \@namedef{bbl@elname@nil}{nil}
7791 \@namedef{bbl@lname@nil}{nil}
7792 \@namedef{bbl@esname@nil}{Latin}
7793 \@namedef{bbl@sname@nil}{Latin}
7794 \@namedef{bbl@sbc@nil}{Latn}
7795 \@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.

```

7796 \ldf@finish{nil}
7797 \</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.

```

7798 <<*Compute Julian day>> ≡
7799 \def\bbl@fpmo#1#2{(#1-#2*floor(#1/#2))}
7800 \def\bbl@cs@gregleap#1{%
7801   (\bbl@fpmo{#1}{4} == 0) &&
7802   (!((\bbl@fpmo{#1}{100} == 0) && (\bbl@fpmo{#1}{400} != 0)))}
7803 \def\bbl@cs@jd#1#2#3{% year, month, day
7804   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7805     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7806     floor((#1 - 1) / 400) + floor(((367 * #2) - 362) / 12) +
7807     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3 } }
7808 <</Compute Julian day>>

```

12.1 Islamic

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

```

7809 <*ca-islamic>
7810 \ExplSyntaxOn
7811 <<Compute Julian day>>
7812 % == islamic (default)
7813 % Not yet implemented
7814 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{}

```

The Civil calendar:

```

7815 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7816   ((#3 + ceil(29.5 * (#2 - 1)) +
7817     (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7818     1948439.5) - 1) }
7819 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7820 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7821 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7822 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7823 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7824 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7825   \edef\bbl@tempa{%
7826     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7827   \edef#5{%
7828     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7829   \edef#6{\fp_eval:n{
7830     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7831   \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).

```

7832 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7833 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7834 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7835 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7836 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7837 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7838 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7839 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7840 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7841 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7842 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7843 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7844 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7845 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7846 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7847 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7848 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7849 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7850 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7851 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7852 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7853 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7854 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7855 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7856 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7857 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7858 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7859 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7860 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7861 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7862 65401,65431,65460,65490,65520}
7863 \@namedef\bbl@ca@islamic-umalqura+{\bbl@ca@islamcuqr@x{+1}}
7864 \@namedef\bbl@ca@islamic-umalqura{\bbl@ca@islamcuqr@x{}}
7865 \@namedef\bbl@ca@islamic-umalqura-{\bbl@ca@islamcuqr@x{-1}}
7866 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7867 \ifnum#2>2014 \ifnum#2<2038
7868 \bbl@afterfi\expandafter\@gobble
7869 \fi\fi
7870 {\bbl@error{Year~out-of~range}{The~allowed~range~is~2014-2038}}%
7871 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7872 \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7873 \count@\@ne
7874 \bbl@foreach\bbl@cs@umalqura@data{%
7875 \advance\count@\@ne
7876 \ifnum##1>\bbl@tempd\else
7877 \edef\bbl@tempe{\the\count@}%
7878 \edef\bbl@tempb{##1}%
7879 \fi}%
7880 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7881 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7882 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
7883 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7884 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
7885 \ExplSyntaxOff
7886 \bbl@add\bbl@precalendar{%
7887 \bbl@replace\bbl@ld@calendar{-civil}{}%
7888 \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7889 \bbl@replace\bbl@ld@calendar{+}{}%
7890 \bbl@replace\bbl@ld@calendar{-}{%

```

7891 \langle /ca-islamic \rangle

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 `hebcsl.sty`

```
7892  $\langle$ *ca-hebrew $\rangle$ 
7893 \newcount\bbl@cntcommon
7894 \def\bbl@remainder#1#2#3{%
7895   #3=#1\relax
7896   \divide #3 by #2\relax
7897   \multiply #3 by -#2\relax
7898   \advance #3 by #1\relax}%
7899 \newif\ifbbl@divisible
7900 \def\bbl@checkifdivisible#1#2{%
7901   {\countdef\tmp=0
7902    \bbl@remainder{#1}{#2}{\tmp}%
7903    \ifnum \tmp=0
7904      \global\bbl@divisibletrue
7905    \else
7906      \global\bbl@divisiblefalse
7907    \fi}}
7908 \newif\ifbbl@gregleap
7909 \def\bbl@ifgregleap#1{%
7910   \bbl@checkifdivisible{#1}{4}%
7911   \ifbbl@divisible
7912     \bbl@checkifdivisible{#1}{100}%
7913     \ifbbl@divisible
7914       \bbl@checkifdivisible{#1}{400}%
7915       \ifbbl@divisible
7916         \bbl@gregleaptrue
7917       \else
7918         \bbl@gregleapfalse
7919       \fi
7920     \else
7921       \bbl@gregleaptrue
7922     \fi
7923   \else
7924     \bbl@gregleapfalse
7925   \fi
7926   \ifbbl@gregleap}
7927 \def\bbl@gregdayspriormonths#1#2#3{%
7928   {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7929     181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7930   \bbl@ifgregleap{#2}%
7931   \ifnum #1 > 2
7932     \advance #3 by 1
7933   \fi
7934   \fi
7935   \global\bbl@cntcommon=#3}%
7936   #3=\bbl@cntcommon}
7937 \def\bbl@gregdaysprioryears#1#2{%
7938   {\countdef\tmpc=4
7939    \countdef\tmpb=2
7940    \tmpb=#1\relax
7941    \advance \tmpb by -1
7942    \tmpc=\tmpb
7943    \multiply \tmpc by 365
7944    #2=\tmpc
7945    \tmpc=\tmpb
7946    \divide \tmpc by 4
7947    \advance #2 by \tmpc
```

```

7948 \tmpc=\tmpb
7949 \divide \tmpc by 100
7950 \advance #2 by -\tmpc
7951 \tmpc=\tmpb
7952 \divide \tmpc by 400
7953 \advance #2 by \tmpc
7954 \global\bbl@cntcommon=#2\relax}%
7955 #2=\bbl@cntcommon}
7956 \def\bbl@absfromgreg#1#2#3#4{%
7957 {\countdef\tmpd=0
7958 #4=#1\relax
7959 \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7960 \advance #4 by \tmpd
7961 \bbl@gregdaysprioryears{#3}{\tmpd}%
7962 \advance #4 by \tmpd
7963 \global\bbl@cntcommon=#4\relax}%
7964 #4=\bbl@cntcommon}
7965 \newif\ifbbl@hebrleap
7966 \def\bbl@checkleaphebryear#1{%
7967 {\countdef\tmpa=0
7968 \countdef\tmpb=1
7969 \tmpa=#1\relax
7970 \multiply \tmpa by 7
7971 \advance \tmpa by 1
7972 \bbl@remainder{\tmpa}{19}{\tmpb}%
7973 \ifnum \tmpb < 7
7974 \global\bbl@hebrleaptrue
7975 \else
7976 \global\bbl@hebrleapfalse
7977 \fi}}
7978 \def\bbl@hebreleapsedmonths#1#2{%
7979 {\countdef\tmpa=0
7980 \countdef\tmpb=1
7981 \countdef\tmpc=2
7982 \tmpa=#1\relax
7983 \advance \tmpa by -1
7984 #2=\tmpa
7985 \divide #2 by 19
7986 \multiply #2 by 235
7987 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7988 \tmpc=\tmpb
7989 \multiply \tmpb by 12
7990 \advance #2 by \tmpb
7991 \multiply \tmpc by 7
7992 \advance \tmpc by 1
7993 \divide \tmpc by 19
7994 \advance #2 by \tmpc
7995 \global\bbl@cntcommon=#2}%
7996 #2=\bbl@cntcommon}
7997 \def\bbl@hebreleapseddays#1#2{%
7998 {\countdef\tmpa=0
7999 \countdef\tmpb=1
8000 \countdef\tmpc=2
8001 \bbl@hebreleapsedmonths{#1}{#2}%
8002 \tmpa=#2\relax
8003 \multiply \tmpa by 13753
8004 \advance \tmpa by 5604
8005 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8006 \divide \tmpa by 25920
8007 \multiply #2 by 29
8008 \advance #2 by 1
8009 \advance #2 by \tmpa
8010 \bbl@remainder{#2}{7}{\tmpa}%

```

```

8011 \ifnum \tmpc < 19440
8012     \ifnum \tmpc < 9924
8013     \else
8014         \ifnum \tmpa=2
8015             \bbl@checkleaphebrewyear{#1}% of a common year
8016             \ifbbl@hebrleap
8017             \else
8018                 \advance #2 by 1
8019             \fi
8020         \fi
8021     \fi
8022     \ifnum \tmpc < 16789
8023     \else
8024         \ifnum \tmpa=1
8025             \advance #1 by -1
8026             \bbl@checkleaphebrewyear{#1}% at the end of leap year
8027             \ifbbl@hebrleap
8028                 \advance #2 by 1
8029             \fi
8030         \fi
8031     \fi
8032 \else
8033     \advance #2 by 1
8034 \fi
8035 \bbl@remainder{#2}{7}{\tmpa}%
8036 \ifnum \tmpa=0
8037     \advance #2 by 1
8038 \else
8039     \ifnum \tmpa=3
8040         \advance #2 by 1
8041     \else
8042         \ifnum \tmpa=5
8043             \advance #2 by 1
8044         \fi
8045     \fi
8046 \fi
8047 \global\bbl@cntcommon=#2\relax}%
8048 #2=\bbl@cntcommon}
8049 \def\bbl@daysinhebrewyear#1#2{%
8050 {\countdef\tmpe=12
8051 \bbl@hebreleapseddays{#1}{\tmpe}%
8052 \advance #1 by 1
8053 \bbl@hebreleapseddays{#1}{#2}%
8054 \advance #2 by -\tmpe
8055 \global\bbl@cntcommon=#2}%
8056 #2=\bbl@cntcommon}
8057 \def\bbl@hebrdayspriormonths#1#2#3{%
8058 {\countdef\tmpf= 14
8059 #3=\ifcase #1\relax
8060     0 \or
8061     0 \or
8062     30 \or
8063     59 \or
8064     89 \or
8065     118 \or
8066     148 \or
8067     148 \or
8068     177 \or
8069     207 \or
8070     236 \or
8071     266 \or
8072     295 \or
8073     325 \or

```



```

8074         400
8075     \fi
8076     \bbl@checkleaphebryear{#2}%
8077     \ifbbl@hebrleap
8078         \ifnum #1 > 6
8079             \advance #3 by 30
8080         \fi
8081     \fi
8082     \bbl@daysinhebryear{#2}{\tmpf}%
8083     \ifnum #1 > 3
8084         \ifnum \tmpf=353
8085             \advance #3 by -1
8086         \fi
8087         \ifnum \tmpf=383
8088             \advance #3 by -1
8089         \fi
8090     \fi
8091     \ifnum #1 > 2
8092         \ifnum \tmpf=355
8093             \advance #3 by 1
8094         \fi
8095         \ifnum \tmpf=385
8096             \advance #3 by 1
8097         \fi
8098     \fi
8099     \global\bbl@cntcommon=#3\relax}%
8100     #3=\bbl@cntcommon}
8101 \def\bbl@absfromhebr#1#2#3#4{%
8102     {#4=#1\relax
8103     \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8104     \advance #4 by #1\relax
8105     \bbl@hebrrelapseddays{#3}{#1}%
8106     \advance #4 by #1\relax
8107     \advance #4 by -1373429
8108     \global\bbl@cntcommon=#4\relax}%
8109     #4=\bbl@cntcommon}
8110 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8111     {\countdef\tmpx= 17
8112     \countdef\tmpy= 18
8113     \countdef\tmpz= 19
8114     #6=#3\relax
8115     \global\advance #6 by 3761
8116     \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8117     \tmpz=1 \tmpy=1
8118     \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8119     \ifnum \tmpx > #4\relax
8120         \global\advance #6 by -1
8121         \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8122     \fi
8123     \advance #4 by -\tmpx
8124     \advance #4 by 1
8125     #5=#4\relax
8126     \divide #5 by 30
8127     \loop
8128         \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8129         \ifnum \tmpx < #4\relax
8130             \advance #5 by 1
8131             \tmpy=\tmpx
8132         \repeat
8133     \global\advance #5 by -1
8134     \global\advance #4 by -\tmpy}}
8135 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8136 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear

```

```

8137 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8138   \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8139   \bbl@hebrfromgreg
8140   {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8141   {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8142   \edef#4{\the\bbl@hebryear}%
8143   \edef#5{\the\bbl@hebrmonth}%
8144   \edef#6{\the\bbl@hebrday}}
8145 \</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).

```

8146 \<ca-persian>
8147 \ExplSyntaxOn
8148 \<<Compute Julian day>>
8149 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8150   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8151 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8152   \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8153   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8154     \bbl@afterfi\expandafter\@gobble
8155   \fi\fi
8156   {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}}%
8157   \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8158   \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8159   \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8160   \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8161   \ifnum\bbl@tempc<\bbl@tempb
8162     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8163     \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8164     \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8165     \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8166   \fi
8167   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8168   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8169   \edef#5{\fp_eval:n{% set Jalali month
8170     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8171   \edef#6{\fp_eval:n{% set Jalali day
8172     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}%
8173 \ExplSyntaxOff
8174 \</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.

```

8175 \<ca-coptic>
8176 \ExplSyntaxOn
8177 \<<Compute Julian day>>
8178 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8179   \edef\bbl@tempd{\fp_eval:n{\floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8180   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8181   \edef#4{\fp_eval:n{%
8182     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8183   \edef\bbl@tempc{\fp_eval:n{%
8184     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8185   \edef#5{\fp_eval:n{\floor(\bbl@tempc / 30) + 1}}%

```

```

8186 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8187 \ExplSyntaxOff
8188 \ca-coptic
8189 *ca-ethiopic
8190 \ExplSyntaxOn
8191 <<Compute Julian day>>
8192 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8193 \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8194 \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8195 \edef#4{\fp_eval:n{%
8196 floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8197 \edef\bbl@tempc{\fp_eval:n{%
8198 \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8199 \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8200 \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}
8201 \ExplSyntaxOff
8202 \ca-ethiopic

```

12.5 Buddhist

That's very simple.

```

8203 *ca-buddhist
8204 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8205 \edef#4{\number\numexpr#1+543\relax}%
8206 \edef#5{#2}%
8207 \edef#6{#3}}
8208 \ca-buddhist
8209 %
8210 % \subsection{Chinese}
8211 %
8212 % Brute force, with the Julian day of first day of each month. The
8213 % table has been computed with the help of \textsf{python-lunardate} by
8214 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8215 % is 2015-2044.
8216 %
8217 % \begin{macrocode}
8218 *ca-chinese
8219 \ExplSyntaxOn
8220 <<Compute Julian day>>
8221 \def\bbl@ca@chinese#1-#2-#3\@@#4#5#6{%
8222 \edef\bbl@tempd{\fp_eval:n{%
8223 \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8224 \count@ \z@
8225 \@tempcnta=2015
8226 \bbl@foreach\bbl@cs@chinese@data{%
8227 \ifnum##1>\bbl@tempd\else
8228 \advance\count@\@ne
8229 \ifnum\count@>12
8230 \count@\@ne
8231 \advance\@tempcnta\@ne\fi
8232 \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8233 \ifin@
8234 \advance\count@\m@ne
8235 \edef\bbl@tempe{\the\numexpr\count@+12\relax}%
8236 \else
8237 \edef\bbl@tempe{\the\count@}%
8238 \fi
8239 \edef\bbl@tempb{##1}%
8240 \fi}%
8241 \edef#4{\the\@tempcnta}%
8242 \edef#5{\bbl@tempe}%
8243 \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8244 \def\bbl@cs@chinese@leap{%

```

```

8245 885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8246 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8247 354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8248 768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8249 1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8250 1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8251 1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8252 2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8253 2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8254 2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8255 3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8256 3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8257 3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8258 4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8259 4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8260 5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8261 5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8262 5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8263 6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8264 6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8265 6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8266 7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8267 7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8268 7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8269 8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8270 8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8271 8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8272 9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8273 9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8274 10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8275 10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8276 10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8277 10896,10926,10956,10986,11015,11045,11074,11103}
8278 \ExplSyntaxOff
8279 </ca-chinese>

```

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 `iniTEX` sees, we need to set some category codes just to be able to change the definition of `\input`.

```

8280 (*bplain | blplain)
8281 \catcode`\{=1 % left brace is begin-group character
8282 \catcode`\}=2 % right brace is end-group character
8283 \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).

```

8284 \openin 0 hyphen.cfg
8285 \ifeof0
8286 \else
8287   \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.

```

8288 \def\input #1 {%
8289   \let\input\input
8290   \a hyphen.cfg
8291   \let\input\undefined
8292 }
8293 \fi
8294 \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`.

```

8295 \bplain)\a plain.tex
8296 \bplain)\a lplain.tex

```

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

```

8297 \bplain)\def\fmtname{babel-plain}
8298 \bplain)\def\fmtname{babel-lplain}

```

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

13.2 Emulating some \LaTeX features

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

```

8299 \langle \langle *Emulate LaTeX \rangle \rangle \equiv
8300 \def\@empty{}
8301 \def\loadlocalcfg#1{%
8302   \openin0#1.cfg
8303   \ifeof0
8304     \closein0
8305   \else
8306     \closein0
8307     {\immediate\writel6{*****}%
8308      \immediate\writel6{* Local config file #1.cfg used}%
8309      \immediate\writel6{*}%
8310     }
8311     \input #1.cfg\relax
8312 \fi
8313 \@endofldf}

```

13.3 General tools

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

```

8314 \long\def\@firstofone#1{#1}
8315 \long\def\@firstoftwo#1#2{#1}
8316 \long\def\@secondoftwo#1#2{#2}
8317 \def\@nnil{\@nil}
8318 \def\@gobbletwo#1#2{}
8319 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8320 \def\@star@or@long#1{%
8321   \@ifstar
8322   {\let\@ngrel@x\relax#1}%

```

```

8323 {\let\l@ngrel@x\long#1}}
8324 \let\l@ngrel@x\relax
8325 \def\@car#1#2\@nil{#1}
8326 \def\@cdr#1#2\@nil{#2}
8327 \let\@typeset@protect\relax
8328 \let\protected@edef\edef
8329 \long\def\@gobble#1{}
8330 \edef\@backslashchar{\expandafter\@gobble\string\}
8331 \def\strip@prefix#1>{}
8332 \def\g@addto@macro#1#2{%
8333     \toks@\expandafter{#1#2}%
8334     \xdef#1{\the\toks@}}
8335 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8336 \def\@nameuse#1{\csname #1\endcsname}
8337 \def\@ifundefined#1{%
8338     \expandafter\ifx\csname#1\endcsname\relax
8339     \expandafter\@firstoftwo
8340     \else
8341     \expandafter\@secondoftwo
8342     \fi}
8343 \def\@expandtwoargs#1#2#3{%
8344     \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8345 \def\zap@space#1 #2{%
8346     #1%
8347     \ifx#2\@empty\else\expandafter\zap@space\fi
8348     #2}
8349 \let\bbl@trace\@gobble
8350 \def\bbl@error#1#2{%
8351     \begingroup
8352     \newlinechar=\^^J
8353     \def\{\^^J(babel) }%
8354     \errhelp{#2}\errmessage{\#1}%
8355     \endgroup}
8356 \def\bbl@warning#1{%
8357     \begingroup
8358     \newlinechar=\^^J
8359     \def\{\^^J(babel) }%
8360     \message{\#1}%
8361     \endgroup}
8362 \let\bbl@infowarn\bbl@warning
8363 \def\bbl@info#1{%
8364     \begingroup
8365     \newlinechar=\^^J
8366     \def\{\^^J}%
8367     \wlog{#1}%
8368     \endgroup}

```

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

```

8369 \ifx\@preamblecmds\@undefined
8370     \def\@preamblecmds{}
8371 \fi
8372 \def\@onlypreamble#1{%
8373     \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8374         \@preamblecmds\do#1}}
8375 \@onlypreamble\@onlypreamble

```

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

```

8376 \def\begindocument{%
8377     \@begindocumenthook
8378     \global\let\@begindocumenthook\@undefined
8379     \def\do##1{\global\let##1\@undefined}%
8380     \@preamblecmds
8381     \global\let\do\noexpand}

```

```

8382 \ifx\@begindocumenthook\@undefined
8383   \def\@begindocumenthook{}
8384 \fi
8385 \@onlypreamble\@begindocumenthook
8386 \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`.

```

8387 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8388 \@onlypreamble\AtEndOfPackage
8389 \def\@endofldf{}
8390 \@onlypreamble\@endofldf
8391 \let\bbl@afterlang\@empty
8392 \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.

```

8393 \catcode`\&=\z@
8394 \ifx&\if@files\@undefined
8395   \expandafter\let\csname if@files\expandafter\endcsname
8396     \csname iffalse\endcsname
8397 \fi
8398 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

8399 \def\newcommand{\@star@or@long\new@command}
8400 \def\new@command#1{%
8401   \@testopt{\@newcommand#1}0}
8402 \def\@newcommand#1[#2]{%
8403   \@ifnextchar [{\@xargdef#1[#2]}%
8404     {\@argdef#1[#2]}}
8405 \long\def\@argdef#1[#2]#3{%
8406   \@yargdef#1\@ne{#2}{#3}}
8407 \long\def\@xargdef#1[#2][#3]#4{%
8408   \expandafter\def\expandafter#1\expandafter{%
8409     \expandafter\@protected@testopt\expandafter #1%
8410     \csname\string#1\expandafter\endcsname{#3}}%
8411   \expandafter\@yargdef \csname\string#1\endcsname
8412   \tw@{#2}{#4}}
8413 \long\def\@yargdef#1#2#3{%
8414   \@tempcnta#3\relax
8415   \advance \@tempcnta \@ne
8416   \let\@hash@\relax
8417   \edef\reserved@a{\ifx#2\tw@ [\@hash@]\fi}%
8418   \@tempcntb #2%
8419   \@whilenum\@tempcntb <\@tempcnta
8420   \do{%
8421     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8422     \advance\@tempcntb \@ne}%
8423   \let\@hash@##%
8424   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
8425 \def\providecommand{\@star@or@long\provide@command}
8426 \def\provide@command#1{%
8427   \begingroup
8428     \escapechar\m@ne\xdef\@gtempa{\string#1}%
8429   \endgroup
8430   \expandafter\ifundefined\@gtempa
8431     {\def\reserved@a{\new@command#1}}%
8432     {\let\reserved@a\relax
8433       \def\reserved@a{\new@command\reserved@a}}%
8434   \reserved@a}%
8435 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}

```

```

8436 \def\declare@robustcommand#1{%
8437   \edef\reserved@a{\string#1}%
8438   \def\reserved@b{#1}%
8439   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8440   \edef#1{%
8441     \ifx\reserved@a\reserved@b
8442       \noexpand\x@protect
8443       \noexpand#1%
8444     \fi
8445     \noexpand\protect
8446     \expandafter\noexpand\csname
8447       \expandafter\@gobble\string#1 \endcsname
8448   }%
8449   \expandafter\new@command\csname
8450     \expandafter\@gobble\string#1 \endcsname
8451 }
8452 \def\x@protect#1{%
8453   \ifx\protect\@typeset@protect\else
8454     \@x@protect#1%
8455   \fi
8456 }
8457 \catcode`\&=\z@ % Trick to hide conditionals
8458 \def\@x@protect#1&fi#2#3{&fi\protect#1}

```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```

8459 \def\bbl@tempa{\csname newif\endcsname&fin@}
8460 \catcode`\&=4
8461 \ifx\in@\@undefined
8462   \def\in@#1#2{%
8463     \def\in@##1#1##2##3\in@{%
8464       \ifx\in@##2\in@false\else\in@true\fi}%
8465     \in@@#2#1\in@\in@@}
8466 \else
8467   \let\bbl@tempa\@empty
8468 \fi
8469 \bbl@tempa

```

\LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain \TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```

8470 \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.

```

8471 \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 $\text{\LaTeX} 2_{\epsilon}$ versions; just enough to make things work in plain \TeX environments.

```

8472 \ifx\@tempcnta\@undefined
8473   \csname newcount\endcsname\@tempcnta\relax
8474 \fi
8475 \ifx\@tempcntb\@undefined
8476   \csname newcount\endcsname\@tempcntb\relax
8477 \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`).

```

8478 \ifx\bye\@undefined
8479   \advance\count10 by -2\relax

```



```

8480 \fi
8481 \ifx\@ifnextchar\@undefined
8482   \def\@ifnextchar#1#2#3{%
8483     \let\reserved@d=#1%
8484     \def\reserved@a{#2}\def\reserved@b{#3}%
8485     \futurelet\@let@token\@ifnch}
8486 \def\@ifnch{%
8487   \ifx\@let@token\@sptoken
8488     \let\reserved@c\@xifnch
8489   \else
8490     \ifx\@let@token\reserved@d
8491       \let\reserved@c\reserved@a
8492     \else
8493       \let\reserved@c\reserved@b
8494     \fi
8495   \fi
8496   \reserved@c}
8497 \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
8498 \def\:{\@xifnch} \expandafter\def\:{ {\futurelet\@let@token\@ifnch}
8499 \fi
8500 \def\@testopt#1#2{%
8501   \@ifnextchar[{\#1}{\#1[\#2]}}
8502 \def\@protected@testopt#1{%
8503   \ifx\protect\@typeset@protect
8504     \expandafter\@testopt
8505   \else
8506     \@x@protect#1%
8507   \fi}
8508 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8509   #2\relax}\fi}
8510 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8511   \else\expandafter\@gobble\fi{#1}}

```

13.4 Encoding related macros

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

```

8512 \def\DeclareTextCommand{%
8513   \@dec@text@cmd\providecommand
8514 }
8515 \def\ProvideTextCommand{%
8516   \@dec@text@cmd\providecommand
8517 }
8518 \def\DeclareTextSymbol#1#2#3{%
8519   \@dec@text@cmd\chardef#1{#2}#3\relax
8520 }
8521 \def\@dec@text@cmd#1#2#3{%
8522   \expandafter\def\expandafter#2%
8523     \expandafter{%
8524       \csname#3-cmd\expandafter\endcsname
8525       \expandafter#2%
8526       \csname#3\string#2\endcsname
8527     }%
8528 %   \let\@ifdefinable\rc@ifdefinable
8529   \expandafter#1\csname#3\string#2\endcsname
8530 }
8531 \def\@current@cmd#1{%
8532   \ifx\protect\@typeset@protect\else
8533     \noexpand#1\expandafter\@gobble
8534   \fi
8535 }
8536 \def\@changed@cmd#1#2{%
8537   \ifx\protect\@typeset@protect
8538     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax

```

```

8539 \expandafter\ifx\csname ?\string#1\endcsname\relax
8540 \expandafter\def\csname ?\string#1\endcsname{%
8541 \@changed@x@err{#1}%
8542 }%
8543 \fi
8544 \global\expandafter\let
8545 \csname\cf@encoding\string#1\expandafter\endcsname
8546 \csname ?\string#1\endcsname
8547 \fi
8548 \csname\cf@encoding\string#1%
8549 \expandafter\endcsname
8550 \else
8551 \noexpand#1%
8552 \fi
8553 }
8554 \def\@changed@x@err#1{%
8555 \errhelp{Your command will be ignored, type <return> to proceed}%
8556 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8557 \def\DeclareTextCommandDefault#1{%
8558 \DeclareTextCommand#1?%
8559 }
8560 \def\ProvideTextCommandDefault#1{%
8561 \ProvideTextCommand#1?%
8562 }
8563 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8564 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8565 \def\DeclareTextAccent#1#2#3{%
8566 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8567 }
8568 \def\DeclareTextCompositeCommand#1#2#3#4{%
8569 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8570 \edef\reserved@b{\string##1}%
8571 \edef\reserved@c{%
8572 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8573 \ifx\reserved@b\reserved@c
8574 \expandafter\expandafter\expandafter\ifx
8575 \expandafter\@car\reserved@a\relax\relax\@nil
8576 \@text@composite
8577 \else
8578 \edef\reserved@b##1{%
8579 \def\expandafter\noexpand
8580 \csname#2\string#1\endcsname###1{%
8581 \noexpand\@text@composite
8582 \expandafter\noexpand\csname#2\string#1\endcsname
8583 ###1\noexpand\@empty\noexpand\@text@composite
8584 {##1}%
8585 }%
8586 }%
8587 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8588 \fi
8589 \expandafter\def\csname\expandafter\string\csname
8590 #2\endcsname\string#1-\string#3\endcsname{#4}
8591 \else
8592 \errhelp{Your command will be ignored, type <return> to proceed}%
8593 \errmessage{\string\DeclareTextCompositeCommand\space used on
8594 inappropriate command \protect#1}
8595 \fi
8596 }
8597 \def\@text@composite#1#2#3\@text@composite{%
8598 \expandafter\@text@composite@x
8599 \csname\string#1-\string#2\endcsname
8600 }
8601 \def\@text@composite@x#1#2{%

```

```

8602 \ifx#1\relax
8603 #2%
8604 \else
8605 #1%
8606 \fi
8607 }
8608 %
8609 \def\@strip@args#1:#2-#3\@strip@args{#2}
8610 \def\DeclareTextComposite#1#2#3#4{%
8611 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8612 \bgroup
8613 \lccode`\@=#4%
8614 \lowercase{%
8615 \egroup
8616 \reserved@a @%
8617 }%
8618 }
8619 %
8620 \def\UseTextSymbol#1#2{#2}
8621 \def\UseTextAccent#1#2#3{}
8622 \def\@use@text@encoding#1{}
8623 \def\DeclareTextSymbolDefault#1#2{%
8624 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8625 }
8626 \def\DeclareTextAccentDefault#1#2{%
8627 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8628 }
8629 \def\cf@encoding{OT1}

```

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

```

8630 \DeclareTextAccent{"}{OT1}{127}
8631 \DeclareTextAccent{'}{OT1}{19}
8632 \DeclareTextAccent{^}{OT1}{94}
8633 \DeclareTextAccent{\`}{OT1}{18}
8634 \DeclareTextAccent{\~}{OT1}{126}

```

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

```

8635 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8636 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8637 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
8638 \DeclareTextSymbol{\textquoteright}{OT1}{``'}
8639 \DeclareTextSymbol{\i}{OT1}{16}
8640 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the \TeX -control sequence `\scriptsize` to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \TeX has, we just `\let` it to `\sevenrm`.

```

8641 \ifx\scriptsize\@undefined
8642 \let\scriptsize\sevenrm
8643 \fi

```

And a few more “dummy” definitions.

```

8644 \def\language{english}%
8645 \let\bbl@opt@shorthands\@nnil
8646 \def\bbl@ifshorthand#1#2#3{#2}%
8647 \let\bbl@language@opts\@empty
8648 \let\bbl@ensureinfo\@gobble
8649 \let\bbl@provide@locale\relax
8650 \ifx\babeloptionstrings\@undefined
8651 \let\bbl@opt@strings\@nnil
8652 \else
8653 \let\bbl@opt@strings\babeloptionstrings
8654 \fi
8655 \def\BabelStringsDefault{generic}

```

```

8656 \def\bbl@tempa{normal}
8657 \ifx\babeloptionmath\bbl@tempa
8658   \def\bbl@mathnormal{\noexpand\textormath}
8659 \fi
8660 \def\AfterBabelLanguage#1#2{}
8661 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8662 \let\bbl@afterlang\relax
8663 \def\bbl@opt@safe{BR}
8664 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8665 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8666 \expandafter\newif\csname ifbbl@single\endcsname
8667 \chardef\bbl@bidimode\z@
8668 <</Emulate LaTeX>>

```

A proxy file:

```

8669 <*plain>
8670 \input babel.def
8671 </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 \TeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The \TeX book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport, *\TeX , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in: \TeX hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018.
- [10] Hubert Partl, *German \TeX* , *TUGboat* 9 (1988) #1, p. 70–72.
- [11] Joachim Schrod, *International \TeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using \TeX* , Springer, 2002, p. 301–373.
- [13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).