

Babel

Code

Version 3.96.31373
2023/11/09

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode

TeX

pdfTeX

LuaTeX

XeTeX

Contents

1	Identification and loading of required files	3
2	locale directory	3
3	Tools	3
3.1	Multiple languages	7
3.2	The Package File (\LaTeX , <code>babel.sty</code>)	8
3.3	<code>base</code>	9
3.4	<code>key=value</code> options and other general option	10
3.5	Conditional loading of shorthands	11
3.6	Interlude for Plain	13
4	Multiple languages	13
4.1	Selecting the language	15
4.2	Errors	23
4.3	Hooks	25
4.4	Setting up language files	27
4.5	Shorthands	29
4.6	Language attributes	38
4.7	Support for saving macro definitions	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	90
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

10	Support for interchar	103
10.1	Layout	105
10.2	8-bit TeX	106
10.3	LuaTeX	107
10.4	Southeast Asian scripts	113
10.5	CJK line breaking	115
10.6	Arabic justification	117
10.7	Common stuff	121
10.8	Automatic fonts and ids switching	121
10.9	Bidi	127
10.10	Layout	129
10.11	Lua: transforms	137
10.12	Lua: Auto bidi with <code>basic</code> and <code>basic-r</code>	145
11	Data for CJK	156
12	The ‘nil’ language	156
13	Calendars	157
13.1	Islamic	157
13.2	Hebrew	159
13.3	Persian	163
13.4	Coptic and Ethiopic	164
13.5	Buddhist	164
14	Support for Plain TeX (<code>plain.def</code>)	166
14.1	Not renaming <code>hyphen.tex</code>	166
14.2	Emulating some L ^A T _E X features	166
14.3	General tools	167
14.4	Encoding related macros	170
15	Acknowledgements	173

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=3.96.31373>>
2 <<date=2023/11/09>>
```

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 pdf_{La}TeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

3.1 Multiple languages

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

```

199 <<(*Define core switching macros)>> ≡

```



```

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

```

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```

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

```

3.3 base

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

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

```

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

```

3.4 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no modifiers have been given, the former is `\relax`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\@for` or `load keyval`, for example.

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

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

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

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea,

anyway.) The first one processes options which has been declared above or follow the syntax `<key>=<value>`, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

3.5 Conditional loading of shorthands

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

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

3.6 Interlude for Plain

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

```

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

```

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

```

448 \</core>
449 \<package | core>

```

4 Multiple languages

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

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

```

450 \def\bbl@version{\<<version>>}
451 \def\bbl@date{\<<date>>}
452 \<<Define core switching macros>>

```

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

```

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

```

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

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

```

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

```

```

471 \edef\bbl@tempd{\noexpand\ifundefined{\noexpand\bbl@tempe#1}}%
472 \bbl@tempd
473 {\lowercase\expandafter{\bbl@tempd}%
474 {\uppercase\expandafter{\bbl@tempd}%
475 \@empty
476 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
477 \uppercase\expandafter{\bbl@tempd}}}%
478 {\edef\bbl@tempd{\def\noexpand#1{#1}}%
479 \lowercase\expandafter{\bbl@tempd}}}%
480 \@empty
481 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
482 \bbl@tempd
483 \bbl@exp{\bbl@usehooks{language}{\{language\}{#1}}}%
484 \def\bbl@iflanguage#1{%
485 \ifundefined{lanerr#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     \babel@error{For a language to be defined on the fly 'base'\\%
531       is not enough, and the whole package must be\\%
532       loaded. Either delete the 'base' option or\\%
533       request the languages explicitly}%
534     {See the manual for further details.}%
535   \fi
536   \let\babel@auxname\language % Still necessary. TODO
537   \babel@ifunset{\babel@bcp@map@\language}{}% Move uplevel??
538   {\edef\language{\@nameuse{\babel@bcp@map@\language}}}%
539   \ifbabel@bcp@allowed
540     \expandafter\ifx\csname date\language\endcsname\relax
541       \expandafter
542       \babel@bcp@lookup\language-\@empty-\@empty-\@empty\@@
543       \ifx\babel@bcp\relax\else % Returned by \babel@bcp@lookup
544         \edef\language{\babel@bcp@prefix\babel@bcp}%
545         \edef\localename{\babel@bcp@prefix\babel@bcp}%
546         \expandafter\ifx\csname date\language\endcsname\relax
547           \let\babel@initoload\babel@bcp
548           \babel@exp{\babelprovide[\babel@autoload@bcptoptions]{\language}}%
549           \let\babel@initoload\relax
550         \fi
551         \babel@csarg\xdef{bcp@map@\babel@bcp}{\localename}%
552       \fi
553     \fi
554   \fi
555   \expandafter\ifx\csname date\language\endcsname\relax
556     \IfFileExists{babel-\language.tex}%
557     {\babel@exp{\babelprovide[\babel@autoload@options]{\language}}}%
558     {}%
559   \fi}
560 (+core)

```

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

```

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

```

4.1 Selecting the language

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

```

568 \let\babel@select@type\z@
569 \edef\selectlanguage{%
570   \noexpand\protect
571   \expandafter\noexpand\csname selectlanguage \endcsname}

```

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

```

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

```

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

```

573 \let\xstring\string

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

The unprotected part of \selectlanguage.

```

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

```

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

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

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

```

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

```

```

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

```

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

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

Then we have to *redefine* `\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras<lang>` command at definition time by expanding the `\csname` primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

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

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

```

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     \csname noextras#1\endcsname
692     \let\originalTeX\@empty
693     \babel@beginsave}%
694   \bbl@usehooks{afterreset}}}%
695   \languageshorthands{none}%
696   % set the locale id

```

```

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

```

```

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

```

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

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

```

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

```

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

```

776 \long\def\endotherlanguage{%
777 \global\@ignoretrue\ignorespaces}

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

It also sets hyphenation exceptions, but only once, because they are global (here language `\lccode's` has been set, too). `\bbl@hyphenation@` is set to relax until the very first `\babelhyphenation`, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that `:ENC` is

taken into account) has been set, then use `\hyphenation` with both global and language exceptions and empty the latter to mark they must not be set again.

```

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

```

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

```

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

```

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\(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@#1@#3}}}%
1030 \bbl@cs{ev@#2@#1@#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}%
1076 {\bbl@exp{%
1077 \\\DeclareRobustCommand<bbl@ensure@language>[1]{%
1078 \\\foreignlanguage{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}%
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\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     }
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 `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`. The `\@gobble` is needed to remove a token such as `\activechar`: (when the double colon was the active character to be dealt with). There are two definitions, depending of `\ifincsn` is available. If there is, the expansion will be more robust.

```

1333 \begingroup
1334 \bbl@ifunset{ifincsn}% 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     \ifincsn
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 \langle lang\rangle\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\curname\string?\string##1\endcurname}%
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\curname###1\string##1\endcurname}}}%
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\@gobblesetwo\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.

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

```

2200 \def\umlauthigh{%
2201   \def\bbl@umlauta##1{\leavevmode\bgroup%
2202     \accent\csname\f@encoding dpos\endcsname
2203     ##1\bbl@allowhyphens\egroup}%
2204   \let\bbl@umlaute\bbl@umlauta}
2205 \def\umlautlow{%
2206   \def\bbl@umlauta{\protect\lower@umlaut}}
2207 \def\umlautelow{%
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 $\langle\textit{dimen}\rangle$ register.

```

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

```

The following code fools `lATEX`'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\f@encoding dqpos\endcsname}%
2218       \dimen@ -.45ex\advance\dimen@ \ht\z@
2219       \ifdim lex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2220     \accent\csname\f@encoding 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@s{#1}}%
2250       {\@dblarg{\bbl@presec@x{#1}}}}}%
2251 \def\bbl@presec@x#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@s#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\name}%
2407 {\bbl@exp{%
2408   \\DeclareRobustCommand\<bbl@ensure@\language\name>[1]{%
2409     \\foreignlanguage{\language\name}%
2410     {####1}}}%
2411 }%
2412 \bbl@exp{%
2413   \\bbl@tglobal\<bbl@ensure@\language\name>%
2414   \\bbl@tglobal\<bbl@ensure@\language\name\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\name}{}%
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\name}}}%
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] = Babel.script_blocks['\bbl@cl{sbcpr}']
2461         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language\name}\space
2462       end
2463     }%

```

```

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

```

```

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

```

```

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

```

```

2653 \catcode\@=11\relax
2654 \def\CurrentOption{#2}%
2655 \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2656 \catcode\@=\atcatcode
2657 \let\atcatcode\relax
2658 \global\bbl@csarg\let{rqtex@\language}\relax
2659 \fi}%
2660 \bbl@foreach\bbl@calendars{%
2661 \bbl@ifunset{\bbl@ca@##1}{%
2662 \chardef\atcatcode=\catcode\@
2663 \catcode\@=11\relax
2664 \InputIfFileExists{babel-ca-##1.tex}{}}}%
2665 \catcode\@=\atcatcode
2666 \let\atcatcode\relax}%
2667 {}}%
2668 \fi
2669 % == frenchspacing ==
2670 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2671 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2672 \ifin@
2673 \bbl@extras@wrap{\bbl@pre@fs}%
2674 {\bbl@pre@fs}%
2675 {\bbl@post@fs}%
2676 \fi
2677 % == transforms ==
2678 % > luababel.def
2679 % == main ==
2680 \ifx\bbl@KVP@main\@nnil % Restore only if not 'main'
2681 \let\language\bbl@savelangname
2682 \chardef\localeid\bbl@savelocaleid\relax
2683 \fi
2684 % == hyphenrules (apply if current) ==
2685 \ifx\bbl@KVP@hyphenrules\@nnil\else
2686 \ifnum\bbl@savelocaleid=\localeid
2687 \language\@nameuse{l@\language}%
2688 \fi
2689 \fi}

```

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

```

2690 \def\bbl@provide@new#1{%
2691 \@namedef{date#1}{}}% marks lang exists - required by \StartBabelCommands
2692 \@namedef{extras#1}{}}%
2693 \@namedef{noextras#1}{}}%
2694 \bbl@startcommands*{#1}{captions}%
2695 \ifx\bbl@KVP@captions\@nnil % and also if import, implicit
2696 \def\bbl@tempb##1% elt for \bbl@captionslist
2697 \ifx##1\@empty\else
2698 \bbl@exp{%
2699 \SetString\##1{%
2700 \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2701 \expandafter\bbl@tempb
2702 \fi}%
2703 \expandafter\bbl@tempb\bbl@captionslist\@empty
2704 \else
2705 \ifx\bbl@initoload\relax
2706 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2707 \else
2708 \bbl@read@ini{\bbl@initoload}2% % Same
2709 \fi
2710 \fi
2711 \StartBabelCommands*{#1}{date}%
2712 \ifx\bbl@KVP@date\@nnil

```

```

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

```

2748 \def\bbl@load@basic#1{%
2749 \ifcase\bbl@howloaded\or\or
2750 \ifcase\csname bbl@llevel@\language\endcsname
2751 \bbl@csarg\let\lname@\language\relax
2752 \fi
2753 \fi
2754 \bbl@ifunset{bbl@lname@#1}%
2755 {\def\BabelBeforeIni##1##2{%
2756 \begingroup
2757 \let\bbl@ini@captions@aux\@gobbletwo
2758 \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
2759 \bbl@read@ini{##1}1%
2760 \ifx\bbl@initoload\relax\endinput\fi
2761 \endgroup}%
2762 \begingroup % boxed, to avoid extra spaces:
2763 \ifx\bbl@initoload\relax
2764 \bbl@input@texini{##1}%
2765 \else
2766 \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2767 \fi
2768 \endgroup}%
2769 {}}

```

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`.

```

2770 \def\bbl@provide@hyphens#1{%
2771   \@tempcnta\m@ne % a flag
2772   \ifx\bbl@KVP@hyphenrules\@nnil\else
2773     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2774     \bbl@foreach\bbl@KVP@hyphenrules{%
2775       \ifnum\@tempcnta=\m@ne % if not yet found
2776         \bbl@ifsamestring{##1}{+}%
2777         {\bbl@carg\addlanguage{l@##1}}%
2778         {}%
2779         \bbl@ifunset{l@##1}% After a possible +
2780         {}%
2781         {\@tempcnta\@nameuse{l@##1}}%
2782       \fi}%
2783     \ifnum\@tempcnta=\m@ne
2784       \bbl@warning{%
2785         Requested 'hyphenrules' for '\language' not found:\\%
2786         \bbl@KVP@hyphenrules.\\%
2787         Using the default value. Reported}%
2788     \fi
2789   \fi
2790   \ifnum\@tempcnta=\m@ne % if no opt or no language in opt found
2791     \ifx\bbl@KVP@captions@\@nnil % TODO. Hackish. See above.
2792       \bbl@ifunset{\bbl@hyphr@#1}{}% use value in ini, if exists
2793       {\bbl@exp{\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2794        {}%
2795        {\bbl@ifunset{l@\bbl@cl{hyphr}}}%
2796        {}% if hyphenrules found:
2797        {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}%
2798     \fi
2799   \fi
2800   \bbl@ifunset{l@#1}%
2801   {\ifnum\@tempcnta=\m@ne
2802     \bbl@carg\adddialect{l@#1}\language
2803     \else
2804     \bbl@carg\adddialect{l@#1}\@tempcnta
2805     \fi}%
2806   {\ifnum\@tempcnta=\m@ne\else
2807     \global\bbl@carg\chardef{l@#1}\@tempcnta
2808     \fi}}

```

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

```

2809 \def\bbl@input@texini#1{%
2810   \bbl@bsphack
2811   \bbl@exp{%
2812     \catcode`\\=14 \catcode`\\=0
2813     \catcode`\\{=1 \catcode`\\}=2
2814     \lowercase{\InputIfFileExists{babel-#1.tex}{}}%
2815     \catcode`\\=\the\catcode`\% \relax
2816     \catcode`\\=\the\catcode`\% \relax
2817     \catcode`\\{=\the\catcode`\{ \relax
2818     \catcode`\\}=\the\catcode`\} \relax}%
2819   \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.

```

2820 \def\bbl@inline#1\bbl@inline{%
2821   \@ifnextchar[\bbl@inisect{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@% ]
2822   \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2823   \def\bbl@iniskip#1\@@{% if starts with ;
2824     \def\bbl@inistore#1=#2\@@{% full (default)
2825     \bbl@trim@def\bbl@tempa{#1}%
2826     \bbl@trim\toks@{#2}%
2827     \bbl@xin@{\bbl@section/\bbl@tempa;\bbl@key@list}%

```



```

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

```

2844 \def\bbl@loop@ini{%
2845 \loop
2846 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2847 \endlinechar\m@ne
2848 \read\bbl@readstream to \bbl@line
2849 \endlinechar\^^M
2850 \ifx\bbl@line\empty\else
2851 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2852 \fi
2853 \repeat}
2854 \ifx\bbl@readstream\undefined
2855 \csname newread\endcsname\bbl@readstream
2856 \fi
2857 \def\bbl@read@ini#1#2{%
2858 \global\let\bbl@extend@ini@gobble
2859 \openin\bbl@readstream=babel-#1.ini
2860 \ifeof\bbl@readstream
2861 \bbl@error
2862 {There is no ini file for the requested language\\%
2863 (#1: \language). Perhaps you misspelled it or your\\%
2864 installation is not complete.}%
2865 {Fix the name or reinstall babel.}%
2866 \else
2867 % == Store ini data in \bbl@inidata ==
2868 \catcode\ [=12 \catcode\]=12 \catcode\==12 \catcode\&=12
2869 \catcode\;=12 \catcode\|=12 \catcode\%=14 \catcode\-=12
2870 \bbl@info{Importing
2871 \ifcase#2font and identification \or basic \fi
2872 data for \language\\%
2873 from babel-#1.ini. Reported}%
2874 \ifnum#2=\z@
2875 \global\let\bbl@inidata\empty
2876 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2877 \fi
2878 \def\bbl@section{identification}%
2879 \bbl@exp{\\bbl@inistore tag.ini=#1\\@@}%
2880 \bbl@inistore load.level=#2\@@
2881 \bbl@loop@ini
2882 % == Process stored data ==
2883 \bbl@csarg\xdef{lini@language}{#1}%

```

```

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

```

2908 \def\bbl@extend@ini@aux#1{%
2909 \bbl@startcommands*{#1}{captions}%
2910 % Activate captions/... and modify exports
2911 \bbl@csarg\def{inikv@captions.licr}##1##2{%
2912 \setlocalecaption{#1}{##1}{##2}}%
2913 \def\bbl@inikv@captions##1##2{%
2914 \bbl@ini@captions@aux{##1}{##2}}%
2915 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2916 \def\bbl@exportkey##1##2##3{%
2917 \bbl@ifunset{bbl@kv@##2}{}%
2918 {\expandafter\ifx\csname bbl@kv@##2\endcsname\empty\else
2919 \bbl@exp{\global\let<bbl@##1@\languagename>\<bbl@kv@##2>}}%
2920 \fi}}%
2921 % As with \bbl@read@ini, but with some changes
2922 \bbl@read@ini@aux
2923 \bbl@ini@exports\tw@
2924 % Update inidata@lang by pretending the ini is read.
2925 \def\bbl@elt##1##2##3{%
2926 \def\bbl@section{##1}%
2927 \bbl@inline##2=##3\bbl@inline}%
2928 \csname bbl@inidata@#1\endcsname
2929 \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2930 \StartBabelCommands*{#1}{date}% And from the import stuff
2931 \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2932 \bbl@savetoday
2933 \bbl@savestate
2934 \bbl@endcommands}

```

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

```

2935 \def\bbl@ini@calendar#1{%
2936 \lowercase{\def\bbl@tempa{=##1=}}%
2937 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2938 \bbl@replace\bbl@tempa{=date.}{}%
2939 \in{.licr}{#1=}%
2940 \ifin@
2941 \ifcase\bbl@engine
2942 \bbl@replace\bbl@tempa{.licr}{}%

```

```

2943 \else
2944 \let\bbl@tempa\relax
2945 \fi
2946 \fi
2947 \ifx\bbl@tempa\relax\else
2948 \bbl@replace\bbl@tempa{=}{}%
2949 \ifx\bbl@tempa\empty\else
2950 \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2951 \fi
2952 \bbl@exp{%
2953 \def<\bbl@inikv@#1>####1####2{%
2954 \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2955 \fi}

```

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

```

2956 \def\bbl@renewinikey#1/#2\@#3{%
2957 \edef\bbl@tempa{\zap@space #1 \@empty}% section
2958 \edef\bbl@tempb{\zap@space #2 \@empty}% key
2959 \bbl@trim\toks@{#3}% value
2960 \bbl@exp{%
2961 \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2962 \\g@addto@macro\\bbl@inidata{%
2963 \\\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

2964 \def\bbl@exportkey#1#2#3{%
2965 \bbl@ifunset{\bbl@kv@#2}%
2966 {\bbl@csarg\gdef{#1@\language\name}{#3}}%
2967 {\expandafter\ifx\csname\bbl@kv@#2\endcsname\empty
2968 \bbl@csarg\gdef{#1@\language\name}{#3}%
2969 \else
2970 \bbl@exp{\global\let<\bbl@#1@\language\name>\<\bbl@kv@#2>}%
2971 \fi}}

```

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

```

2972 \def\bbl@iniwarning#1{%
2973 \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2974 {\bbl@warning{%
2975 From babel-\bbl@cs{lini@\language\name}.ini:\\%
2976 \bbl@cs{@kv@identification.warning#1}\\%
2977 Reported }}}
2978 %
2979 \let\bbl@release@transforms\@empty
2980 \def\bbl@ini@exports#1{%
2981 % Identification always exported
2982 \bbl@iniwarning}%
2983 \ifcase\bbl@engine
2984 \bbl@iniwarning{.pdf\latex}%
2985 \or
2986 \bbl@iniwarning{.lua\latex}%
2987 \or
2988 \bbl@iniwarning{.xel\latex}%
2989 \fi%
2990 \bbl@exportkey{llevel}{identification.load.level}{}%
2991 \bbl@exportkey{elname}{identification.name.english}{}%

```

```

2992 \bbl@exp{\bbl@exportkey{lname}{identification.name.opentype}%
2993   {\csname bbl@lname@\language\endcsname}}%
2994 \bbl@exportkey{tbc}{identification.tag.bcp47}}}%
2995 % Somewhat hackish. TODO
2996 \bbl@exportkey{casing}{identification.tag.bcp47}}}%
2997 \bbl@exportkey{lbc}{identification.language.tag.bcp47}}}%
2998 \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2999 \bbl@exportkey{esname}{identification.script.name}}}%
3000 \bbl@exp{\bbl@exportkey{sname}{identification.script.name.opentype}%
3001   {\csname bbl@esname@\language\endcsname}}%
3002 \bbl@exportkey{sbc}{identification.script.tag.bcp47}}}%
3003 \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3004 \bbl@exportkey{rbcp}{identification.region.tag.bcp47}}}%
3005 \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}}}%
3006 \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}}}%
3007 \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}}}%
3008 \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}}}%
3009 % Also maps bcp47 -> language
3010 \ifbbl@bcptoname
3011   \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
3012 \fi
3013 \ifcase\bbl@engine\or
3014   \directlua{%
3015     Babel.locale_props[\the\bbl@cs{id@\language}].script
3016     = '\bbl@cl{sbc}}}%
3017 \fi
3018 % Conditional
3019 \ifnum#1>\z@      % 0 = only info, 1, 2 = basic, (re)new
3020   \bbl@exportkey{calpr}{date.calendar.preferred}}}%
3021   \bbl@exportkey{lbrk}{typography.linebreaking}{h}%
3022   \bbl@exportkey{hyphr}{typography.hyphenrules}}}%
3023   \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3024   \bbl@exportkey{rgtm}{typography.righthyphenmin}{3}%
3025   \bbl@exportkey{prehc}{typography.prehyphenchar}}}%
3026   \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}}}%
3027   \bbl@exportkey{hyots}{typography.hyphenate.other.script}}}%
3028   \bbl@exportkey{intsp}{typography.intraspaces}}}%
3029   \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3030   \bbl@exportkey{chrng}{characters.ranges}}}%
3031   \bbl@exportkey{quote}{characters.delimiters.quotes}}}%
3032   \bbl@exportkey{dgnat}{numbers.digits.native}}}%
3033   \ifnum#1=\tw@      % only (re)new
3034     \bbl@exportkey{rqtex}{identification.require.babel}}}%
3035     \bbl@tglobal\bbl@savetoday
3036     \bbl@tglobal\bbl@savestate
3037     \bbl@savestrings
3038   \fi
3039 \fi}

```

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

```

3040 \def\bbl@inikv#1#2{%      key=value
3041   \toks@{#2}%             This hides #'s from ini values
3042   \bbl@csarg\xdef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3043 \let\bbl@inikv@identification\bbl@inikv
3044 \let\bbl@inikv@date\bbl@inikv
3045 \let\bbl@inikv@typography\bbl@inikv
3046 \let\bbl@inikv@characters\bbl@inikv
3047 \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’.

```

3048 \def\bbl@inikv@counters#1#2{%
3049   \bbl@ifsamestring{#1}{digits}%
3050   {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3051     decimal digits}%
3052     {Use another name.}}%
3053   }%
3054 \def\bbl@tempc{#1}%
3055 \bbl@trim@def{\bbl@tempb*}{#2}%
3056 \in@{.1$}{#1$}%
3057 \ifin@
3058   \bbl@replace\bbl@tempc{.1}{}%
3059   \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}\{
3060     \noexpand\bbl@alphanumeric{\bbl@tempc}}%
3061 \fi
3062 \in@{.F.}{#1}%
3063 \ifin@else\in@{.S.}{#1}\fi
3064 \ifin@
3065   \bbl@csarg\protected@xdef{cntr@#1@\language}\{\bbl@tempb*}%
3066 \else
3067   \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3068   \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3069   \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3070 \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.

```

3071 \ifcase\bbl@engine
3072   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3073     \bbl@ini@captions@aux{#1}{#2}}
3074 \else
3075   \def\bbl@inikv@captions#1#2{%
3076     \bbl@ini@captions@aux{#1}{#2}}
3077 \fi

```

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

```

3078 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3079   \bbl@replace\bbl@tempa{.template}{}%
3080   \def\bbl@toreplace{#1}{}%
3081   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3082   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3083   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3084   \bbl@replace\bbl@toreplace{[ ]}{\name\endcsname}}%
3085   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3086   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3087   \ifin@
3088     \@nameuse{\bbl@patch\bbl@tempa}%
3089     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3090 \fi
3091 \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3092 \ifin@
3093   \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3094   \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3095     \\bbl@ifunset{\bbl@bbl@tempa fmt@\\language}%
3096     {\[fnum@\bbl@tempa]}%
3097     {\[\\@nameuse{\bbl@bbl@tempa fmt@\\language}}}}%
3098 \fi}
3099 \def\bbl@ini@captions@aux#1#2{%
3100   \bbl@trim@def\bbl@tempa{#1}%
3101   \bbl@xin@{.template}{\bbl@tempa}%
3102   \ifin@
3103     \bbl@ini@captions@template{#2}\language
3104   \else
3105     \bbl@ifblank{#2}%

```

```

3106     {\bbl@exp{%
3107         \toks@{\bbl@nocaption{\bbl@tempa}{\language\language\bbl@tempa name}}}%
3108     {\bbl@trim\toks@{#2}}}%
3109 \bbl@exp{%
3110     \bbl@add\bbl@savestrings{%
3111         \SetString\<\bbl@tempa name>{\the\toks@}}}%
3112     \toks@{\expandafter\bbl@captionslist}%
3113     \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3114     \ifin@
3115         \bbl@exp{%
3116             \bbl@add\<\bbl@extracaps@\language\language>{\<\bbl@tempa name>}%
3117             \bbl@tglobal\<\bbl@extracaps@\language\language>}%
3118         \fi
3119     \fi}

```

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

```

3120 \def\bbl@list@the{%
3121     part,chapter,section,subsection,subsubsection,paragraph,%
3122     subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3123     table,page,footnote,mpfootnote,mpfn}
3124 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3125     \bbl@ifunset{\bbl@map@#1@\language\language}%
3126         {\@nameuse{#1}}%
3127         {\@nameuse{\bbl@map@#1@\language\language}}}
3128 \def\bbl@inikv@labels#1#2{%
3129     \in@{.map}{#1}%
3130     \ifin@
3131         \ifx\bbl@KVP@labels\@nnil\else
3132             \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3133             \ifin@
3134                 \def\bbl@tempc{#1}%
3135                 \bbl@replace\bbl@tempc{.map}{}%
3136                 \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3137                 \bbl@exp{%
3138                     \gdef\<\bbl@map@\bbl@tempc @\language\language>%
3139                     {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3140                 \bbl@foreach\bbl@list@the{%
3141                     \bbl@ifunset{\the##1}{}%
3142                     {\bbl@exp{\let\\bbl@tempd\<\the##1>}%
3143                     \bbl@exp{%
3144                         \bbl@sreplace\<\the##1>%
3145                         {\<\bbl@tempc>{##1}}{\bbl@map@cnt{\bbl@tempc}{##1}}%
3146                         \bbl@sreplace\<\the##1>%
3147                         {\<\@empty @\bbl@tempc>\<c@##1>}{\bbl@map@cnt{\bbl@tempc}{##1}}}%
3148                     \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3149                     \toks@{\expandafter\expandafter\expandafter{
3150                         \csname the##1\endcsname}%
3151                     \expandafter\xdef\csname the##1\endcsname{\the\toks@}}%
3152                     \fi}}%
3153             \fi
3154         \fi
3155     %
3156 \else
3157     %
3158     % The following code is still under study. You can test it and make
3159     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3160     % language dependent.
3161     \in@{enumerate.}{#1}%
3162     \ifin@
3163         \def\bbl@tempa{#1}%
3164         \bbl@replace\bbl@tempa{enumerate.}{}%
3165         \def\bbl@toreplace{#2}%
3166         \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%

```

```

3167 \bbl@replace\bbl@toreplace{[]}{\csname the}%
3168 \bbl@replace\bbl@toreplace{[]}{\endcsname{}}%
3169 \toks@ \expandafter{\bbl@toreplace}%
3170 % TODO. Execute only once:
3171 \bbl@exp{%
3172   \\\bbl@add<extras\language>{%
3173     \\\babel@save<labelenum\romannumeral\bbl@tempa>%
3174     \def<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3175   \\\bbl@tglobal\<extras\language>}%
3176 \fi
3177 \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.

```

3178 \def\bbl@chapttype{chapter}
3179 \ifx\@makechapterhead\@undefined
3180 \let\bbl@patchchapter\relax
3181 \else\ifx\thechapter\@undefined
3182 \let\bbl@patchchapter\relax
3183 \else\ifx\ps@headings\@undefined
3184 \let\bbl@patchchapter\relax
3185 \else
3186 \def\bbl@patchchapter{%
3187   \global\let\bbl@patchchapter\relax
3188   \gdef\bbl@chfmt{%
3189     \bbl@ifunset{\bbl@chapttype fmt@\language}%
3190     {\@chapapp\space\thechapter}
3191     {\@nameuse{\bbl@chapttype fmt@\language}}}
3192   \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3193   \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3194   \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3195   \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3196   \bbl@tglobal\appendix
3197   \bbl@tglobal\ps@headings
3198   \bbl@tglobal\chaptermark
3199   \bbl@tglobal\@makechapterhead}
3200 \let\bbl@patchappendix\bbl@patchchapter
3201 \fi\fi\fi
3202 \ifx\@part\@undefined
3203 \let\bbl@patchpart\relax
3204 \else
3205 \def\bbl@patchpart{%
3206   \global\let\bbl@patchpart\relax
3207   \gdef\bbl@partformat{%
3208     \bbl@ifunset{\bbl@partfmt@\language}%
3209     {\partname\nobreakspace\thepart}
3210     {\@nameuse{\bbl@partfmt@\language}}}
3211   \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3212   \bbl@tglobal\@part}
3213 \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

```

3214 \let\bbl@calendar\@empty
3215 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3216 \def\bbl@localedate#1#2#3#4{%
3217   \begingroup
3218     \edef\bbl@they{#2}%
3219     \edef\bbl@them{#3}%
3220     \edef\bbl@thed{#4}%
3221     \edef\bbl@tempe{%
3222       \bbl@ifunset{\bbl@calpr@\language}{\bbl@cl{calpr}},%

```

```

3223     #1}%
3224     \bbl@replace\bbl@tempe{ }{}%
3225     \bbl@replace\bbl@tempe{CONVERT}{convert=% Hackish
3226     \bbl@replace\bbl@tempe{convert}{convert=%}
3227     \let\bbl@ld@calendar@empty
3228     \let\bbl@ld@variant@empty
3229     \let\bbl@ld@convert\relax
3230     \def\bbl@tempb##1=##2\@@{\@namedef\bbl@ld@##1}{##2}}%
3231     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3232     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3233     \ifx\bbl@ld@calendar@empty\else
3234         \ifx\bbl@ld@convert\relax\else
3235             \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3236             {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3237         \fi
3238     \fi
3239     \@nameuse\bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3240     \edef\bbl@calendar{% Used in \month..., too
3241         \bbl@ld@calendar
3242         \ifx\bbl@ld@variant@empty\else
3243             .\bbl@ld@variant
3244         \fi}%
3245     \bbl@cased
3246     {\@nameuse\bbl@date@languagename @\bbl@calendar}%
3247     \bbl@they\bbl@them\bbl@thed}%
3248     \endgroup}
3249 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3250 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3251     \bbl@trim@def\bbl@tempa{#1.#2}%
3252     \bbl@ifsamestring{\bbl@tempa}{months.wide}%           to savedate
3253     {\bbl@trim@def\bbl@tempa{#3}%
3254         \bbl@trim\toks@{#5}%
3255         \@temptokena\expandafter{\bbl@savedate}%
3256         \bbl@exp{% Reverse order - in ini last wins
3257             \def\\bbl@savedate{%
3258                 \\SetString<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3259                 \the@temptokena}}}%
3260     {\bbl@ifsamestring{\bbl@tempa}{date.long}%           defined now
3261         {\lowercase{\def\bbl@tempb{#6}}%
3262             \bbl@trim@def\bbl@toreplace{#5}%
3263             \bbl@TG@@@date
3264             \global\bbl@csarg\let{date@languagename @\bbl@tempb}\bbl@toreplace
3265             \ifx\bbl@savetoday@empty
3266                 \bbl@exp{% TODO. Move to a better place.
3267                     \\AfterBabelCommands{%
3268                         \def<\languagename date>{\protect<\languagename date >}%
3269                         \\newcommand<\languagename date >[4][{}%
3270                             \\bbl@usedategroupttrue
3271                             <\bbl@ensure@languagename>{%
3272                                 \\localdate[####1]{####2}{####3}{####4}}}%
3273                     \def\\bbl@savetoday{%
3274                         \\SetString\\today{%
3275                             <\languagename date>[convert]%
3276                             {\the\year}{\the\month}{\the\day}}}%
3277                 \fi}%
3278             {}}}}

```

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).

```

3279 \let\bbl@calendar@empty

```



```

3280 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3281   \@nameuse{bbl@ca@#2}#1\@@}
3282 \newcommand\BabelDateSpace{\nobreakspace}
3283 \newcommand\BabelDateDot{\. \@} % TODO. \let instead of repeating
3284 \newcommand\BabelDated[1]{\number#1}
3285 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3286 \newcommand\BabelDateM[1]{\number#1}
3287 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3288 \newcommand\BabelDateMMM[1]{%
3289   \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3290 \newcommand\BabelDatey[1]{\number#1}%
3291 \newcommand\BabelDateyy[1]{%
3292   \ifnum#1<10 0\number#1 %
3293   \else\ifnum#1<100 \number#1 %
3294   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3295   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3296   \else
3297     \bbl@error
3298     {Currently two-digit years are restricted to the\
3299     range 0-9999.}%
3300     {There is little you can do. Sorry.}%
3301   \fi\fi\fi\fi}
3302 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3303 \newcommand\BabelDateU[1]{\number#1}%
3304 \def\bbl@replace@finish@iii#1{%
3305   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3306 \def\bbl@TG@date{%
3307   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3308   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3309   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3310   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3311   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3312   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3313   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3314   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3315   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3316   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3317   \bbl@replace\bbl@toreplace{[U]}{\BabelDateU{###1}}%
3318   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecntr{###1|}}%
3319   \bbl@replace\bbl@toreplace{[U]}{\bbl@datecntr{###1|}}%
3320   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecntr{###2|}}%
3321   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecntr{###3|}}%
3322   \bbl@replace@finish@iii\bbl@toreplace}
3323 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3324 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}

```

Transforms.

```

3325 \let\bbl@release@transforms\empty
3326 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3327 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3328 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3329   #1[#2]{#3}{#4}{#5}}
3330 \begingroup % A hack. TODO. Don't require an specific order
3331   \catcode`\%=12
3332   \catcode`\&=14
3333   \gdef\bbl@transforms#1#2#3{%&%
3334     \directlua{
3335       local str = [==[#2]==]
3336       str = str:gsub('%.%d+%.%d+$', '')
3337       token.set_macro('babeltempa', str)
3338     }&%
3339     \def\babeltempc{}&%
3340     \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&%

```

```

3341 \ifin@else
3342 \bbl@xin@{:babeltempa,}{,\bbl@KVP@transforms,}%&
3343 \fi
3344 \ifin@
3345 \bbl@foreach\bbl@KVP@transforms{%&
3346 \bbl@xin@{:babeltempa,}{,##1,}%&
3347 \ifin@ &% font:font:transform syntax
3348 \directlua{
3349     local t = {}
3350     for m in string.gmatch('##1'..' ':'(.-):') do
3351         table.insert(t, m)
3352     end
3353     table.remove(t)
3354     token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3355 }&%
3356 \fi}&%
3357 \in@{.0$}{#2$}&%
3358 \ifin@
3359 \directlua{%& (\attribute) syntax
3360     local str = string.match([[ \bbl@KVP@transforms]],
3361         '%([^(%[]-)%)[^)]-\babeltempa')
3362     if str == nil then
3363         token.set_macro('babeltempb', '')
3364     else
3365         token.set_macro('babeltempb', ',attribute=' .. str)
3366     end
3367 }&%
3368 \toks@{#3}&%
3369 \bbl@exp{%&
3370     \\g@addto@macro\\bbl@release@transforms{%&
3371     \relax &% Closes previous \bbl@transforms@aux
3372     \\bbl@transforms@aux
3373     \\\#1{label=\babeltempa\babeltempb\babeltempc}&%
3374     {\languagename}{\the\toks@}}&%
3375 \else
3376 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3377 \fi
3378 \fi}
3379 \endgroup

```

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

```

3380 \def\bbl@provide@lsys#1{%
3381 \bbl@ifunset{bbl@lname@#1}%
3382 {\bbl@load@info{#1}}%
3383 }%
3384 \bbl@csarg\let{lsys@#1}\@empty
3385 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3386 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3387 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3388 \bbl@ifunset{bbl@lname@#1}{%
3389 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3390 \ifcase\bbl@engine\or\or
3391 \bbl@ifunset{bbl@prehc@#1}{%
3392 {\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3393 }%
3394 {\ifx\bbl@xenoxyph\undefined
3395 \global\let\bbl@xenoxyph\bbl@xenoxyph@d
3396 \ifx\AtBeginDocument\@notprerr
3397 \expandafter\@secondoftwo % to execute right now
3398 \fi
3399 \AtBeginDocument{%
3400 \bbl@patchfont{\bbl@xenoxyph}%

```

```

3401         \expandafter\select@language\expandafter{\language}%
3402     \fi}%
3403 \fi
3404 \bbl@csarg\bbl@tglobal{lsys@#1}}
3405 \def\bbl@xeno-hyph@d{%
3406     \bbl@ifset{\bbl@prehc@language}%
3407     {\ifnum\hyphenchar\font=\default-hyphenchar
3408         \iffontchar\font\bbl@c{prehc}\relax
3409         \hyphenchar\font\bbl@c{prehc}\relax
3410         \else\iffontchar\font"200B
3411             \hyphenchar\font"200B
3412         \else
3413             \bbl@warning
3414             {Neither 0 nor ZERO WIDTH SPACE are available\\%
3415              in the current font, and therefore the hyphen\\%
3416              will be printed. Try changing the fontspec's\\%
3417              'HyphenChar' to another value, but be aware\\%
3418              this setting is not safe (see the manual).\\%
3419              Reported}%
3420             \hyphenchar\font\default-hyphenchar
3421         \fi\fi
3422     \fi}%
3423     {\hyphenchar\font\default-hyphenchar}}
3424 % \fi}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3425 \def\bbl@load@info#1{%
3426     \def\BabelBeforeIni##1##2{%
3427         \begin{group}
3428             \bbl@read@ini{##1}0%
3429             \endinput           % babel- .tex may contain only preamble's
3430             \endgroup}%         boxed, to avoid extra spaces:
3431     {\bbl@input@texini{#1}}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T_EX. Non-digits characters are kept. The first macro is the generic “localized” command.

```

3432 \def\bbl@setdigits#1#2#3#4#5{%
3433     \bbl@exp{%
3434         \def<\language digits>####1{%           ie, \langdigits
3435             \<\bbl@digits@language>####1\\@nil}%
3436             \let<\bbl@cnt@digits@language>\<\language digits>%
3437             \def<\language counter>####1{%       ie, \langcounter
3438                 \\\expandafter\<\bbl@counter@language>%
3439                 \\\csname c@####1\endcsname}%
3440             \def<\bbl@counter@language>####1{% ie, \bbl@counter@lang
3441                 \\\expandafter\<\bbl@digits@language>%
3442                 \\\number####1\\@nil}}}%
3443     \def\bbl@tempa##1##2##3##4##5{%
3444         \bbl@exp{%           Wow, quite a lot of hashes! :- (
3445             \def<\bbl@digits@language>#####1{%
3446                 \\\ifx#####1\\@nil           % ie, \bbl@digits@lang
3447                 \\\else
3448                     \\\ifx0#####1#1%
3449                     \\\else\\\ifx1#####1#2%
3450                     \\\else\\\ifx2#####1#3%
3451                     \\\else\\\ifx3#####1#4%
3452                     \\\else\\\ifx4#####1#5%
3453                     \\\else\\\ifx5#####1#1%
3454                     \\\else\\\ifx6#####1#2%
3455                     \\\else\\\ifx7#####1#3%

```

[illegible]

```

3463 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
3464   \ifx\#\1%           % \ before, in case #1 is multiletter
3465     \bbl@exp{%
3466       \def\#\bbl@tempa####1{%
3467         \<ifcase>####1\space\the\toks@\<else>\#\@ctrerr\<fi>}}%
3468     \else
3469       \toks@\expandafter{\the\toks@\or #1}%
3470       \expandafter\bbl@buildifcase
3471     \fi}

```

```

3472 \newcommand\localenatural[2]{\bbl@cs{cntr@#1@\languageName}{#2}}
3473 \def\bbl@localecntr#1#2{\localenatural{#2}{#1}}
3474 \newcommand\localecounter[2]{%
3475   \expandafter\bbl@localecntr
3476   \expandafter{\number\csname c@#2\endcsname}{#1}}
3477 \def\bbl@alphanumeric#1#2{%
3478   \expandafter\bbl@alphanumeric@i\number#2 76543210@@{#1}}
3479 \def\bbl@alphanumeric@i#1#2#3#4#5#6#7#8@@@#9{%
3480   \ifcase\car#8@\nil\or    % Currenty <10000, but prepared for bigger
3481     \bbl@alphanumeric@ii{#9}000000#1\or
3482     \bbl@alphanumeric@ii{#9}00000#1#2\or
3483     \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3484     \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3485     \bbl@alphanum@invalid{>9999}%
3486   \fi}
3487 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3488   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languageName}%
3489   {\bbl@cs{cntr@#1.4@\languageName}#5%
3490    \bbl@cs{cntr@#1.3@\languageName}#6%
3491    \bbl@cs{cntr@#1.2@\languageName}#7%
3492    \bbl@cs{cntr@#1.1@\languageName}#8%
3493    \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3494     \bbl@ifunset{bbl@cntr@#1.S.321@\languageName}{}%
3495     {\bbl@cs{cntr@#1.S.321@\languageName}}%
3496   \fi}%
3497   {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languageName}}%
3498 \def\bbl@alphanum@invalid#1{%
3499   \bbl@error{Alphabetic numeral too large (#1)}%
3500   {Currently this is the limit.}}

```

```

3501 \def\bbl@localeinfo#1#2{%
3502   \bbl@ifunset{\bbl@info@#2}{#1}%
3503     {\bbl@ifunset{\bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3504       {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3505 \newcommand\localeinfo[1]{%
3506   \ifx*#1@empty    % TODO. A bit hackish to make it expandable.
3507     \bbl@afterelse\bbl@localeinfo}%
3508   \else

```

```

3509 \bbl@localeinfo
3510 {\bbl@error{I've found no info for the current locale.\\%
3511 The corresponding ini file has not been loaded\\%
3512 Perhaps it doesn't exist}%
3513 {See the manual for details.}}%
3514 {#1}%
3515 \fi}
3516 % \@namedef{bbl@info@name.locale}{lcname}
3517 \@namedef{bbl@info@tag.ini}{lini}
3518 \@namedef{bbl@info@name.english}{elname}
3519 \@namedef{bbl@info@name.opentype}{lname}
3520 \@namedef{bbl@info@tag.bcp47}{tbc}
3521 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3522 \@namedef{bbl@info@tag.opentype}{lotf}
3523 \@namedef{bbl@info@script.name}{esname}
3524 \@namedef{bbl@info@script.name.opentype}{sname}
3525 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3526 \@namedef{bbl@info@script.tag.opentype}{sotf}
3527 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3528 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3529 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3530 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3531 \@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.

```

3532 \providecommand\BCPdata{}
3533 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3534 \renewcommand\BCPdata[1]{\bbl@bcpdata@i#1\@empty}
3535 \def\bbl@bcpdata@i#1#2#3#4#5#6\@empty{%
3536 \nameuse{str_if_eq:nnTF}{#1#2#3#4#5}{main.}%
3537 {\bbl@bcpdata@ii{#6}\bbl@main@language}%
3538 {\bbl@bcpdata@ii{#1#2#3#4#5#6}\language}}%
3539 \def\bbl@bcpdata@ii#1#2{%
3540 \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3541 {\bbl@error{Unknown field '#1' in \string\BCPdata.\\%
3542 Perhaps you misspelled it.}%
3543 {See the manual for details.}}%
3544 {\bbl@ifunset{bbl@csname bbl@info@#1.tag.bcp47\endcsname @#2}{}%
3545 {\bbl@cs{csname bbl@info@#1.tag.bcp47\endcsname @#2}}}%
3546 \fi
3547 % Still somewhat hackish. WIP.
3548 \@namedef{bbl@info@casing.tag.bcp47}{casing}
3549 \newcommand\BabelUppercaseMapping[3]{%
3550 \let\bbl@temp\language
3551 \edef\language{#1}%
3552 \DeclareUppercaseMapping[\BCPdata{casing}]{#2}{#3}%
3553 \let\language\bbl@temp}
3554 \newcommand\BabelLowercaseMapping[3]{%
3555 \let\bbl@temp\language
3556 \edef\language{#1}%
3557 \DeclareLowercaseMapping[\BCPdata{casing}]{#2}{#3}%
3558 \let\language\bbl@temp}

```

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

```

3559 <<*More package options>> \equiv
3560 \DeclareOption{ensureinfo=off}{}
3561 <</More package options>>
3562 \let\bbl@ensureinfo\@gobble
3563 \newcommand\BabelEnsureInfo{%
3564 \ifx\InputIfFileExists\@undefined\else
3565 \def\bbl@ensureinfo##1{%
3566 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%

```

```

3567 \fi
3568 \bbl@foreach\bbl@loaded{{%
3569   \let\bbl@ensuring\@empty % Flag used in a couple of babel-*.tex files
3570   \def\language{##1}%
3571   \bbl@ensureinfo{##1}}}%
3572 \@ifpackagewith{babel}{ensureinfo=off}}}%
3573 {\AtEndOfPackage{% Test for plain.
3574   \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`.

```

3575 \newcommand\getlocaleproperty{%
3576   \ifstar\bbl@getproperty@s\bbl@getproperty@x}
3577 \def\bbl@getproperty@s#1#2#3{%
3578   \let#1\relax
3579   \def\bbl@elt##1##2##3{%
3580     \bbl@ifsamestring{##1/##2}{##3}%
3581     {\providecommand#1{##3}%
3582     \def\bbl@elt###1####2####3{}}}%
3583   {}}%
3584   \bbl@cs{inidata@#2}}}%
3585 \def\bbl@getproperty@x#1#2#3{%
3586   \bbl@getproperty@s{#1}{#2}{#3}%
3587   \ifx#1\relax
3588     \bbl@error
3589     {Unknown key for locale '#2':\%
3590      #3\}%
3591     \string#1 will be set to \relax}%
3592     {Perhaps you misspelled it.}%
3593   \fi}
3594 \let\bbl@ini@loaded\@empty
3595 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

5 Adjusting the Babel bahavior

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

```

3596 \newcommand\babeladjust[1]{% TODO. Error handling.
3597   \bbl@forkv{#1}{%
3598     \bbl@ifunset{\bbl@ADJ@##1##2}%
3599     {\bbl@cs{ADJ@##1}{##2}}%
3600     {\bbl@cs{ADJ@##1@##2}}}%
3601 %
3602 \def\bbl@adjust@lua#1#2{%
3603   \ifvmode
3604     \ifnum\currentgrouplevel=\z@
3605       \directlua{ Babel.#2 }%
3606       \expandafter\expandafter\expandafter\@gobble
3607     \fi
3608   \fi
3609   {\bbl@error % The error is gobbled if everything went ok.
3610    {Currently, #1 related features can be adjusted only\%
3611     in the main vertical list.}%
3612    {Maybe things change in the future, but this is what it is.}}}
3613 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3614   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3615 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3616   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3617 \@namedef{\bbl@ADJ@bidi.text@on}{%
3618   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3619 \@namedef{\bbl@ADJ@bidi.text@off}{%
3620   \bbl@adjust@lua{bidi}{bidi_enabled=false}}

```

```

3621 \@namedef{bbl@ADJ@bidi.math@on}{%
3622   \let\bbl@noamsmath\@empty}
3623 \@namedef{bbl@ADJ@bidi.math@off}{%
3624   \let\bbl@noamsmath\relax}
3625 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3626   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3627 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3628   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3629 %
3630 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3631   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3632 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3633   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3634 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3635   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3636 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3637   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3638 \@namedef{bbl@ADJ@justify.arabic@on}{%
3639   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3640 \@namedef{bbl@ADJ@justify.arabic@off}{%
3641   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3642 %
3643 \def\bbl@adjust@layout#1{%
3644   \ifvmode
3645     #1%
3646     \expandafter\@gobble
3647   \fi
3648   {\bbl@error   % The error is gobbled if everything went ok.
3649     {Currently, layout related features can be adjusted only\\%
3650       in vertical mode.}%
3651     {Maybe things change in the future, but this is what it is.}}}
3652 \@namedef{bbl@ADJ@layout.tabular@on}{%
3653   \ifnum\bbl@tabular@mode=\tw@
3654     \bbl@adjust@layout{\let\@tabular\bbl@NL@@@tabular}%
3655   \else
3656     \chardef\bbl@tabular@mode\@ne
3657   \fi}
3658 \@namedef{bbl@ADJ@layout.tabular@off}{%
3659   \ifnum\bbl@tabular@mode=\tw@
3660     \bbl@adjust@layout{\let\@tabular\bbl@OL@@@tabular}%
3661   \else
3662     \chardef\bbl@tabular@mode\z@
3663   \fi}
3664 \@namedef{bbl@ADJ@layout.lists@on}{%
3665   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3666 \@namedef{bbl@ADJ@layout.lists@off}{%
3667   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3668 %
3669 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3670   \bbl@bcpallowedtrue}
3671 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3672   \bbl@bcpallowedfalse}
3673 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3674   \def\bbl@bcp@prefix{#1}}
3675 \def\bbl@bcp@prefix{bcp47-}
3676 \@namedef{bbl@ADJ@autoload.options}#1{%
3677   \def\bbl@autoload@options{#1}}
3678 \let\bbl@autoload@bcptoptions\@empty
3679 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3680   \def\bbl@autoload@bcptoptions{#1}}
3681 \newif\ifbbl@bcptoname
3682 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3683   \bbl@bcptonametrue

```

```

3684 \BabelEnsureInfo}
3685 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3686 \bbl@bcp47tonamefalse}
3687 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3688 \directlua{ Babel.ignore_pre_char = function(node)
3689 return (node.lang == \the\csname l@nohyphenation\endcsname)
3690 end }}
3691 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3692 \directlua{ Babel.ignore_pre_char = function(node)
3693 return false
3694 end }}
3695 \@namedef{bbl@ADJ@select.write@shift}{%
3696 \let\bbl@restorelastskip\relax
3697 \def\bbl@savelastskip{%
3698 \let\bbl@restorelastskip\relax
3699 \ifvmode
3700 \ifdim\lastskip=\z@
3701 \let\bbl@restorelastskip\nobreak
3702 \else
3703 \bbl@exp{%
3704 \def\\bbl@restorelastskip{%
3705 \skip@=\the\lastskip
3706 \\nobreak \vskip-\skip@ \vskip\skip@}}%
3707 \fi
3708 \fi}}
3709 \@namedef{bbl@ADJ@select.write@keep}{%
3710 \let\bbl@restorelastskip\relax
3711 \let\bbl@savelastskip\relax}
3712 \@namedef{bbl@ADJ@select.write@omit}{%
3713 \AddBabelHook{babel-select}{beforestart}{%
3714 \expandafter\babel@aux\expandafter\bbl@main@language}}}%
3715 \let\bbl@restorelastskip\relax
3716 \def\bbl@savelastskip##1\bbl@restorelastskip{}%
3717 \@namedef{bbl@ADJ@select.encoding@off}{%
3718 \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.

```

3719 <<More package options>> ≡
3720 \DeclareOption{safe=none}{\let\bbl@opt@safe\empty}
3721 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3722 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3723 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3724 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3725 <</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.

```

3726 \bbl@trace{Cross referencing macros}
3727 \ifx\bbl@opt@safe\empty\else % ie, if 'ref' and/or 'bib'
3728 \def\@newl@bel#1#2#3{%
3729 {\@safe@activestrue

```



```

3730 \bbl@ifunset{#1@#2}%
3731 \relax
3732 {\gdef\@multiplelabels{%
3733 \@latex@warning@no@line{There were multiply-defined labels}}}%
3734 \@latex@warning@no@line{Label `#2' multiply defined}}}%
3735 \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.

```

3736 \CheckCommand*\@testdef[3]{%
3737 \def\reserved@a{#3}%
3738 \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3739 \else
3740 \@tempswatrue
3741 \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.

```

3742 \def\@testdef#1#2#3{% TODO. With @samestring?
3743 \@safe@activetrue
3744 \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3745 \def\bbl@tempb{#3}%
3746 \@safe@activetrue
3747 \ifx\bbl@tempa\relax
3748 \else
3749 \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3750 \fi
3751 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3752 \ifx\bbl@tempa\bbl@tempb
3753 \else
3754 \@tempswatrue
3755 \fi}
3756 \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.

```

3757 \bbl@xin@{R}\bbl@opt@safe
3758 \ifin@
3759 \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3760 \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3761 {\expandafter\strip@prefix\meaning\ref}%
3762 \ifin@
3763 \bbl@redefine\@kernel@ref#1{%
3764 \@safe@activetrue\org@@kernel@ref{#1}\@safe@activetrue}
3765 \bbl@redefine\@kernel@pageref#1{%
3766 \@safe@activetrue\org@@kernel@pageref{#1}\@safe@activetrue}
3767 \bbl@redefine\@kernel@sref#1{%
3768 \@safe@activetrue\org@@kernel@sref{#1}\@safe@activetrue}
3769 \bbl@redefine\@kernel@spageref#1{%
3770 \@safe@activetrue\org@@kernel@spageref{#1}\@safe@activetrue}
3771 \else
3772 \bbl@redefineroobust\ref#1{%
3773 \@safe@activetrue\org@ref{#1}\@safe@activetrue}
3774 \bbl@redefineroobust\pageref#1{%
3775 \@safe@activetrue\org@pageref{#1}\@safe@activetrue}
3776 \fi
3777 \else
3778 \let\org@ref\ref
3779 \let\org@pageref\pageref
3780 \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.

```
3781 \bbl@xin@{B}\bbl@opt@safe
3782 \ifin@
3783 \bbl@redefine\@citex[#1]#2{%
3784   \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3785   \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.

```
3786 \AtBeginDocument{%
3787   \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.)

```
3788   \def\@citex[#1][#2]#3{%
3789     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3790     \org@@citex[#1][#2]{\@tempa}}%
3791   }{}}
```

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

```
3792 \AtBeginDocument{%
3793   \ifpackageloaded{cite}{%
3794     \def\@citex[#1]#2{%
3795       \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
3796     }{}}
```

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

```
3797 \bbl@redefine\nocite#1{%
3798   \@safe@activetrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3799 \bbl@redefine\bibcite{%
3800   \bbl@cite@choice
3801   \bibcite}
```

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

```
3802 \def\bbl@bibcite#1#2{%
3803   \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.

```
3804 \def\bbl@cite@choice{%
3805   \global\let\bibcite\bbl@bibcite
3806   \ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3807   \ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3808   \global\let\bbl@cite@choice\relax}
```

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

```
3809 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3810 \bbl@redefine\@bibitem#1{%
3811   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
3812 \else
3813   \let\org@nocite\nocite
3814   \let\org@citex\@citex
3815   \let\org@bibtex\@bibtex
3816   \let\org@bibitem\@bibitem
3817 \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.

```
3818 \bbl@trace{Marks}
3819 \IfBabelLayout{sectioning}
3820   {\ifx\bbl@opt@headfoot\@nnil
3821     \g@addto@macro\@resetactivechars{%
3822       \set@typeset@protect
3823       \expandafter\select@language\expandafter{\bbl@main@language}%
3824       \let\protect\noexpand
3825       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3826         \edef\thepage{%
3827           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3828       \fi}%
3829   \fi}
3830 {\ifbbl@single\else
3831   \bbl@ifunset{markright }\bbl@redefine\bbl@redefineroobust
3832   \markright#1{%
3833     \bbl@ifblank{#1}%
3834     {\org@markright{}}%
3835     {\toks@{#1}%
3836       \bbl@exp{%
3837         \\org@markright{\\protect\\foreignlanguage{\language}\language}%
3838         {\\protect\\bbl@restore@actives\the\toks@}}}%
3839   }
```

\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.)

```
3839   \ifx\@mkboth\markboth
3840     \def\bbl@tempc{\let\@mkboth\markboth}%
3841   \else
3842     \def\bbl@tempc{%
3843       \fi
3844       \bbl@ifunset{markboth }\bbl@redefine\bbl@redefineroobust
3845       \markboth#1#2{%
3846         \protected@edef\bbl@tempb##1{%
3847           \protect\foreignlanguage
3848             {\language}\protect\bbl@restore@actives##1}%
3849         \bbl@ifblank{#1}%
3850         {\toks@{}}%
```

```

3851      {\toks@expandafter{\bbl@tempb{#1}}}%
3852      \bbl@ifblank{#2}%
3853      {\@temptokena{}}%
3854      {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3855      \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}}%
3856      \bbl@tempc
3857      \fi} % end ifbbl@single, end \IfBabelLayout

```

5.3 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.

```

3858 \bbl@trace{Preventing clashes with other packages}
3859 \ifx\org@ref\@undefined\else
3860   \bbl@xin@{R}\bbl@opt@safe
3861   \ifin@
3862     \AtBeginDocument{%
3863       \ifpackageloaded{ifthen}{%
3864         \bbl@redefine@long\ifthenelse#1#2#3{%
3865           \let\bbl@temp@pref\pageref
3866           \let\pageref\org@pageref
3867           \let\bbl@temp@ref\ref
3868           \let\ref\org@ref
3869           \@safe@activestrue
3870           \org@ifthenelse{#1}%
3871             {\let\pageref\bbl@temp@pref
3872              \let\ref\bbl@temp@ref
3873              \@safe@activesfalse
3874              #2}%
3875             {\let\pageref\bbl@temp@pref
3876              \let\ref\bbl@temp@ref
3877              \@safe@activesfalse
3878              #3}%
3879           }%
3880         }{}%
3881       }
3882 \fi

```

5.3.2 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`.

```

3883 \AtBeginDocument{%
3884   \@ifpackageloaded{varioref}{%
3885     \bbl@redefine\@vppageref#1[#2]#3{%
3886       \@safe@activestrue

```

```

3887      \org@@vpageref{#1}[#2]{#3}%
3888      \@safe@activesfalse}%
3889      \bbl@redefine\vrefpagenum#1#2{%
3890      \@safe@activestrue
3891      \org@vrefpagenum{#1}{#2}%
3892      \@safe@activesfalse}%

```

The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref_` to call `\org@ref` instead of `\ref`. The disadvantage of this solution is that whenever the definition of `\Ref` changes, this definition needs to be updated as well.

```

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

```

5.3.3 `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.

```

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

```

`\substitutefontfamily` *Deprecated.* 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.

```

3907 \def\substitutefontfamily#1#2#3{%
3908   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3909   \immediate\write15{%
3910     \string\ProvidesFile{#1#2.fd}%
3911     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}]
3912     \space generated font description file^^J
3913     \string\DeclareFontFamily{#1}{#2}{^^J
3914     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
3915     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
3916     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
3917     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
3918     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
3919     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
3920     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
3921     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
3922     }%
3923     \closeout15
3924   }
3925 \onlypreamble\substitutefontfamily

```

5.4 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

```

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

```

Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which latin fontencoding to use.

\latinencoding When text is being typeset in an encoding other than ‘latin’ (OT1 or T1), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

```

3961 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}

```

But this might be overruled with a later loading of the package fontenc. Therefore we check at the execution of \begin{document} whether it was loaded with the T1 option. The normal way to do this (using \@ifpackageloaded) is disabled for this package. Now we have to revert to parsing the internal macro \@filelist which contains all the filenames loaded.

```

3962 \AtBeginDocument{%
3963   \@ifpackageloaded{fontspec}%
3964     {\xdef\latinencoding{%
3965       \ifx\UTFencname\@undefined
3966         EU\ifcase\bbl@engine\or2\or1\fi
3967       \else
3968         \UTFencname
3969       \fi}}%
3970   {\gdef\latinencoding{OT1}%
3971     \ifx\cf@encoding\bbl@t@one
3972       \xdef\latinencoding{\bbl@t@one}%

```

```

3973 \else
3974 \def\@elt#1{,#1,}%
3975 \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3976 \let\@elt\relax
3977 \bbl@xin@{,T1,}\bbl@tempa
3978 \ifin@
3979 \xdef\latinencoding{\bbl@t@one}%
3980 \fi
3981 \fi}}

```

`\latintext` Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

```

3982 \DeclareRobustCommand{\latintext}{%
3983 \fontencoding{\latinencoding}\selectfont
3984 \def\encodingdefault{\latinencoding}}

```

`\textlatin` This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

```

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

```

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

```

3990 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}

```

5.5 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.

```

3991 \bbl@trace{Loading basic (internal) bidi support}
3992 \ifodd\bbl@engine
3993 \else % TODO. Move to txtbabel
3994 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 % Any xe+lua bidi=
3995 \bbl@error
3996 {The bidi method 'basic' is available only in\\%
3997 \luatex. I'll continue with 'bidi=default', so\\%
3998 expect wrong results}%
3999 {See the manual for further details.}%
4000 \let\bbl@beforeforeign\leavevmode
4001 \AtEndOfPackage{%
4002 \EnableBabelHook{babel-bidi}%

```

```

4003     \bbl@xebidipar}
4004 \fi\fi
4005 \def\bbl@loadxebidi#1{%
4006     \ifx\RTLfootnotetext\undefined
4007     \AtEndOfPackage{%
4008         \EnableBabelHook{babel-bidi}%
4009         \bbl@loadfontspec % bidi needs fontspec
4010         \usepackage#1{bidi}%
4011         \let\bbl@digitsdotdash\DigitsDotDashInterCharToks
4012         \def\DigitsDotDashInterCharToks{% See the 'bidi' package
4013             \ifnum\@nameuse{\bbl@wdir\@languagename}=\tw@ % 'AL' bidi
4014                 \bbl@digitsdotdash % So ignore in 'R' bidi
4015             \fi}}%
4016     \fi}
4017 \ifnum\bbl@bidimode>200 % Any xe bidi=
4018     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
4019         \bbl@tentative{bidi=bidi}
4020         \bbl@loadxebidi{}
4021     \or
4022         \bbl@loadxebidi{[rldocument]}
4023     \or
4024         \bbl@loadxebidi{}
4025     \fi
4026 \fi
4027 \fi
4028 % TODO? Separate:
4029 \ifnum\bbl@bidimode=\@ne % Any bidi= except default=1
4030     \let\bbl@beforeforeign\leavevmode
4031     \ifodd\bbl@engine
4032         \newattribute\bbl@attr@dir
4033         \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
4034         \bbl@exp{\output{\bodydir\pagedir\the\output}}
4035     \fi
4036     \AtEndOfPackage{%
4037         \EnableBabelHook{babel-bidi}%
4038         \ifodd\bbl@engine\else
4039             \bbl@xebidipar
4040         \fi}
4041 \fi

```

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

```

4042 \bbl@trace{Macros to switch the text direction}
4043 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4044 \def\bbl@rscripts{% TODO. Base on codes ??
4045     ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4046     Old Hungarian,Lydian,Mandaean,Manichaean,%
4047     Meroitic Cursive,Meroitic,Old North Arabian,%
4048     Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4049     Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4050     Old South Arabian,}%
4051 \def\bbl@provide@dirs#1{%
4052     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4053     \ifin@
4054         \global\bbl@csarg\chardef{wdir@#1}\@ne
4055         \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4056         \ifin@
4057             \global\bbl@csarg\chardef{wdir@#1}\tw@
4058         \fi
4059     \else
4060         \global\bbl@csarg\chardef{wdir@#1}\z@
4061     \fi
4062     \ifodd\bbl@engine

```



```

4063 \bbl@csarg\ifcase{wdir@#1}%
4064 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'l' }%
4065 \or
4066 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'r' }%
4067 \or
4068 \directlua{ Babel.locale_props[\the\localeid].texmdir = 'al' }%
4069 \fi
4070 \fi}
4071 \def\bbl@switchdir{%
4072 \bbl@ifunset{bbl@lsys{\language}\bbl@provide@lsys{\language}}{}%
4073 \bbl@ifunset{bbl@wdir{\language}\bbl@provide@dirs{\language}}{}%
4074 \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}%
4075 \def\bbl@setdirs#1{% TODO - math
4076 \ifcase\bbl@select@type % TODO - strictly, not the right test
4077 \bbl@bodydir{#1}%
4078 \bbl@paddir{#1}% <- Must precede \bbl@texmdir
4079 \fi
4080 \bbl@texmdir{#1}}
4081 % TODO. Only if \bbl@bidimode > 0?:
4082 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4083 \DisableBabelHook{babel-bidi}

```

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

```

4084 \ifodd\bbl@engine % luatex=1
4085 \else % pdftex=0, xetex=2
4086 \newcount\bbl@dirlevel
4087 \chardef\bbl@thetexmdir\z@
4088 \chardef\bbl@thepaddir\z@
4089 \def\bbl@texmdir#1{%
4090 \ifcase#1\relax
4091 \chardef\bbl@thetexmdir\z@
4092 \@nameuse{setlatin}%
4093 \bbl@texmdir@i\beginL\endL
4094 \else
4095 \chardef\bbl@thetexmdir@ne
4096 \@nameuse{setnonlatin}%
4097 \bbl@texmdir@i\beginR\endR
4098 \fi}
4099 \def\bbl@texmdir@i#1#2{%
4100 \ifhmode
4101 \ifnum\currentgrouplevel>\z@
4102 \ifnum\currentgrouplevel=\bbl@dirlevel
4103 \bbl@error{Multiple bidi settings inside a group}%
4104 {I'll insert a new group, but expect wrong results.}%
4105 \bgroup\aftergroup#2\aftergroup\egroup
4106 \else
4107 \ifcase\currentgrouptype\or % 0 bottom
4108 \aftergroup#2% 1 simple {}
4109 \or
4110 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4111 \or
4112 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4113 \or\or\or % vbox vtop align
4114 \or
4115 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4116 \or\or\or\or\or\or % output math disc insert vcent mathchoice
4117 \or
4118 \aftergroup#2% 14 \begingroup
4119 \else
4120 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4121 \fi
4122 \fi
4123 \bbl@dirlevel\currentgrouplevel

```

```

4124 \fi
4125 #1%
4126 \fi}
4127 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4128 \let\bbl@bodydir\@gobble
4129 \let\bbl@pagedir\@gobble
4130 \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).

```

4131 \def\bbl@xebidipar{%
4132 \let\bbl@xebidipar\relax
4133 \TeXeTstate\@ne
4134 \def\bbl@xeverypar{%
4135 \ifcase\bbl@thepardir
4136 \ifcase\bbl@thetextdir\else\beginR\fi
4137 \else
4138 {\setbox\z@\lastbox\beginR\box\z@}%
4139 \fi}%
4140 \let\bbl@severypar\everypar
4141 \newtoks\everypar
4142 \everypar=\bbl@severypar
4143 \bbl@severypar{\bbl@xeverypar\the\everypar}}
4144 \ifnum\bbl@bidimode>200 % Any xe bidi=
4145 \let\bbl@textdir@i\@gobbletwo
4146 \let\bbl@xebidipar\@empty
4147 \AddBabelHook{bidi}{foreign}{%
4148 \def\bbl@tempa{\def\BabelText###1}%
4149 \ifcase\bbl@thetextdir
4150 \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4151 \else
4152 \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4153 \fi}
4154 \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4155 \fi
4156 \fi

```

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

```

4157 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4158 \AtBeginDocument{%
4159 \ifx\pdfstringdefDisableCommands\@undefined\else
4160 \ifx\pdfstringdefDisableCommands\relax\else
4161 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4162 \fi
4163 \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`.

```

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

```

```

4173 \@empty}}
4174 \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).

```

4175 \bbl@trace{Language options}
4176 \let\bbl@afterlang\relax
4177 \let\BabelModifiers\relax
4178 \let\bbl@loaded\@empty
4179 \def\bbl@load@language#1{%
4180   \InputIfFileExists{#1.ldf}%
4181   {\edef\bbl@loaded{\CurrentOption
4182     \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4183     \expandafter\let\expandafter\bbl@afterlang
4184       \csname\CurrentOption.ldf-h@k\endcsname
4185     \expandafter\let\expandafter\BabelModifiers
4186       \csname\bbl@mod@\CurrentOption\endcsname
4187     \bbl@expf{\AtBeginDocument}%
4188     \bbl@usehooks@lang{\CurrentOption}{begindocument}{\CurrentOption}}}%
4189   {\bbl@error{%
4190     Unknown option '\CurrentOption'. Either you misspelled it\\%
4191     or the language definition file \CurrentOption.ldf was not found}%
4192     Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4193     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4194     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.

```

4195 \def\bbl@try@load@lang#1#2#3{%
4196   \IfFileExists{\CurrentOption.ldf}%
4197   {\bbl@load@language{\CurrentOption}}%
4198   {\#1\bbl@load@language{\#2}\#3}}
4199 %
4200 \DeclareOption{hebrew}{%
4201   \input{rlbabel.def}%
4202   \bbl@load@language{hebrew}}
4203 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4204 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4205 \DeclareOption{northernsami}{\bbl@try@load@lang{}{samin}{}}
4206 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4207 \DeclareOption{polutonikogreek}{%
4208   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4209 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4210 \DeclareOption{scottishgaelic}{\bbl@try@load@lang{}{scottish}{}}
4211 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4212 \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.

```

4213 \ifx\bbl@opt@config\@nnil
4214   \@ifpackagewith{babel}{noconfigs}{}%
4215   {\InputIfFileExists{bblopts.cfg}%
4216     {\typeout{*****^J%
4217       * Local config file bblopts.cfg used^^J%
4218       *}}%
4219     {}}%
4220 \else

```

```

4221 \InputIfFileExists{\bbl@opt@config.cfg}%
4222 {\typeout{*****^J%
4223          * Local config file \bbl@opt@config.cfg used^J%
4224          *}}%
4225 {\bbl@error{%
4226   Local config file '\bbl@opt@config.cfg' not found}{%
4227   Perhaps you misspelled it.}}%
4228 \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.

```

4229 \ifx\bbl@opt@main\@nnil
4230 \ifnum\bbl@iniflag>\z@ % if all ldf's: set implicitly, no main pass
4231 \let\bbl@tempb\@empty
4232 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4233 \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4234 \bbl@foreach\bbl@tempb{% \bbl@tempb is a reversed list
4235 \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4236 \ifodd\bbl@iniflag % = * =
4237 \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}}%
4238 \else % n +=
4239 \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}}%
4240 \fi
4241 \fi}%
4242 \fi
4243 \else
4244 \bbl@info{Main language set with 'main='. Except if you have\\%
4245 problems, prefer the default mechanism for setting\\%
4246 the main language, ie, as the last declared.\\%
4247 Reported}
4248 \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`).

```

4249 \ifx\bbl@opt@main\@nnil\else
4250 \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4251 \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4252 \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.

```

4253 \bbl@foreach\bbl@language@opts{%
4254 \def\bbl@tempa{#1}%
4255 \ifx\bbl@tempa\bbl@opt@main\else
4256 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)
4257 \bbl@ifunset{ds@#1}%
4258 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4259 {}%
4260 \else % + * (other = ini)
4261 \DeclareOption{#1}{%
4262 \bbl@ldfinit
4263 \babelprovide[import]{#1}%
4264 \bbl@afterldf{}}%
4265 \fi
4266 \fi}%
4267 \bbl@foreach\@classoptionslist{%
4268 \def\bbl@tempa{#1}%
4269 \ifx\bbl@tempa\bbl@opt@main\else
4270 \ifnum\bbl@iniflag<\tw@ % 0 0 (other = ldf)

```

```

4271 \bbl@ifunset{ds@#1}%
4272 {\IfFileExists{#1.ldf}%
4273 {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4274 {}}%
4275 {}%
4276 \else % + * (other = ini)
4277 \IfFileExists{babel-#1.tex}%
4278 {\DeclareOption{#1}{%
4279 \bbl@ldfinit
4280 \babelprovide[import]{#1}%
4281 \bbl@afterldf{}}}%
4282 {}}%
4283 \fi
4284 \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):

```

4285 \def\AfterBabelLanguage#1{%
4286 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang{}}
4287 \DeclareOption*{}
4288 \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.

```

4289 \bbl@trace{Option 'main'}
4290 \ifx\bbl@opt@main@nnil
4291 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4292 \let\bbl@tempc@empty
4293 \edef\bbl@templ{\bbl@loaded,}
4294 \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4295 \bbl@for\bbl@tempb\bbl@tempa{%
4296 \edef\bbl@tempd{\bbl@tempb,}%
4297 \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4298 \bbl@xin{\bbl@tempd}{\bbl@templ}%
4299 \ifin\edef\bbl@tempc{\bbl@tempb}\fi}
4300 \def\bbl@tempa#1,#2@nnil{\def\bbl@tempb{#1}}
4301 \expandafter\bbl@tempa\bbl@loaded,@nnil
4302 \ifx\bbl@tempb\bbl@tempc\else
4303 \bbl@warning{%
4304 Last declared language option is '\bbl@tempc',\%
4305 but the last processed one was '\bbl@tempb'.\%
4306 The main language can't be set as both a global\%
4307 and a package option. Use 'main=\bbl@tempc' as\%
4308 option. Reported}
4309 \fi
4310 \else
4311 \ifodd\bbl@iniflag % case 1,3 (main is ini)
4312 \bbl@ldfinit
4313 \let\CurrentOption\bbl@opt@main
4314 \bbl@exp{% \bbl@opt@provide = empty if *
4315 \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4316 \bbl@afterldf{}
4317 \DeclareOption{\bbl@opt@main}{}
4318 \else % case 0,2 (main is ldf)
4319 \ifx\bbl@loadmain\relax
4320 \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4321 \else
4322 \DeclareOption{\bbl@opt@main}{\bbl@loadmain}

```

```

4323 \fi
4324 \ExecuteOptions{\bbl@opt@main}
4325 \@namedef{ds@\bbl@opt@main}{}%
4326 \fi
4327 \DeclareOption*{}
4328 \ProcessOptions*
4329 \fi
4330 \bbl@exp{%
4331 \\\AtBeginDocument{\\\bbl@usehooks@lang{}}{begindocument}{}}}%
4332 \def\AfterBabelLanguage{%
4333 \bbl@error
4334 {Too late for \string\AfterBabelLanguage}%
4335 {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.

4336 \ifx\bbl@main@language\@undefined
4337 \bbl@info{%
4338 You haven't specified a language as a class or package\\%
4339 option. I'll load 'nil'. Reported}
4340 \bbl@load@language{nil}
4341 \fi
4342 \</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 T_EX users might want to use some of the features of the babel system too, care has to be taken that plain T_EX 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 T_EX and L^AT_EX, some of it is for the L^AT_EX case only.

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

A proxy file for switch.def

```

4343 \<*kernel>
4344 \let\bbl@onlyswitch\@empty
4345 \input babel.def
4346 \let\bbl@onlyswitch\@undefined
4347 \</kernel>
4348 \<*patterns>

```

7 Loading hyphenation patterns

The following code is meant to be read by iniT_EX because it should instruct T_EX 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.

```

4349 \<<Make sure ProvidesFile is defined>>
4350 \ProvidesFile{hyphen.cfg}[<<date>> v<<version>> Babel hyphens]
4351 \xdef\bbl@format{\jobname}
4352 \def\bbl@version{<<version>>}
4353 \def\bbl@date{<<date>>}
4354 \ifx\AtBeginDocument\@undefined
4355 \def\@empty{}
4356 \fi
4357 \<<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.

```
4358 \def\process@line#1#2 #3 #4 {%
4359   \ifx=#1%
4360     \process@synonym{#2}%
4361   \else
4362     \process@language{#1#2}{#3}{#4}%
4363   \fi
4364   \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.

```
4365 \toks@{}
4366 \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.

```
4367 \def\process@synonym#1{%
4368   \ifnum\last@language=\m@ne
4369     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4370   \else
4371     \expandafter\chardef\csname l@#1\endcsname\last@language
4372     \wlog{\string\l@#1=\string\language\the\last@language}%
4373     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4374       \csname\language\hyphenmins\endcsname
4375     \let\bbl@elt\relax
4376     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}}}%
4377   \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` or `\uccode` arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

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

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

`\bbl@elt{<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.

```
4378 \def\process@language#1#2#3{%
4379   \expandafter\addlanguage\csname l@#1\endcsname
```

```

4380 \expandafter\language\csname l@#1\endcsname
4381 \edef\language{#1}%
4382 \bbl@hook@everylanguage{#1}%
4383 % > luatex
4384 \bbl@get@enc#1::\@@@
4385 \begingroup
4386 \lefthyphenmin\m@ne
4387 \bbl@hook@loadpatterns{#2}%
4388 % > luatex
4389 \ifnum\lefthyphenmin=\m@ne
4390 \else
4391 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4392 \the\lefthyphenmin\the\righthyphenmin}%
4393 \fi
4394 \endgroup
4395 \def\bbl@tempa{#3}%
4396 \ifx\bbl@tempa\@empty\else
4397 \bbl@hook@loadexceptions{#3}%
4398 % > luatex
4399 \fi
4400 \let\bbl@elt\relax
4401 \edef\bbl@languages{%
4402 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4403 \ifnum\the\language=\z@
4404 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4405 \set@hyphenmins\tw@\thr@@\relax
4406 \else
4407 \expandafter\expandafter\expandafter\set@hyphenmins
4408 \csname #1hyphenmins\endcsname
4409 \fi
4410 \the\toks@
4411 \toks@{}%
4412 \fi}

```

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

```

4413 \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.

```

4414 \def\bbl@hook@everylanguage#1{}
4415 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4416 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4417 \def\bbl@hook@loadkernel#1{}
4418 \def\addlanguage{\csname newlanguage\endcsname}%
4419 \def\adddialect##1##2{%
4420 \global\chardef##1##2\relax
4421 \wlog{\string##1 = a dialect from \string\language##2}}%
4422 \def\iflanguage##1{%
4423 \expandafter\ifx\csname l@##1\endcsname\relax
4424 \@nolanerr{##1}%
4425 \else
4426 \ifnum\csname l@##1\endcsname=\language
4427 \expandafter\expandafter\expandafter\@firstoftwo
4428 \else
4429 \expandafter\expandafter\expandafter\@secondoftwo
4430 \fi
4431 \fi}%
4432 \def\providehyphenmins##1##2{%
4433 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4434 \@namedef{##1hyphenmins}{##2}%
4435 \fi}%

```



```

4436 \def\set@hyphenmins##1##2{%
4437   \leftthyphenmin##1\relax
4438   \righthyphenmin##2\relax}%
4439 \def\selectlanguage{%
4440   \errhelp{Selecting a language requires a package supporting it}%
4441   \errmessage{Not loaded}}%
4442 \let\foreignlanguage\selectlanguage
4443 \let\otherlanguage\selectlanguage
4444 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4445 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4446 \def\setlocale{%
4447   \errhelp{Find an armchair, sit down and wait}%
4448   \errmessage{Not yet available}}%
4449 \let\uselocale\setlocale
4450 \let\locale\setlocale
4451 \let\selectlocale\setlocale
4452 \let\localename\setlocale
4453 \let\textlocale\setlocale
4454 \let\textlanguage\setlocale
4455 \let\language\text\setlocale}
4456 \begingroup
4457 \def\AddBabelHook#1#2{%
4458   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4459   \def\next{\toks1}%
4460   \else
4461     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname###1}%
4462     \fi
4463     \next}
4464 \ifx\directlua@\undefined
4465   \ifx\XeTeXinputencoding@\undefined\else
4466     \input xebabel.def
4467     \fi
4468   \else
4469     \input luababel.def
4470     \fi
4471   \openin1 = babel-\bbl@format.cfg
4472   \ifeof1
4473   \else
4474     \input babel-\bbl@format.cfg\relax
4475   \fi
4476   \closein1
4477 \endgroup
4478 \bbl@hook@loadkernel{switch.def}

```

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

```

4479 \openin1 = language.dat

```

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

```

4480 \def\language\name{english}%
4481 \ifeof1
4482   \message{I couldn't find the file language.dat,\space
4483     I will try the file hyphen.tex}
4484   \input hyphen.tex\relax
4485   \chardef\l@english\z@
4486 \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`.

```

4487   \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.

```
4488 \loop
4489   \endlinechar\m@ne
4490   \readl to \bbl@line
4491   \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.

```
4492   \if T\ifeoflF\fi T\relax
4493   \ifx\bbl@line\empty\else
4494     \edef\bbl@line{\bbl@line\space\space\space}%
4495     \expandafter\process@line\bbl@line\relax
4496   \fi
4497 \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.

```
4498 \begingroup
4499   \def\bbl@elt#1#2#3#4{%
4500     \global\language=#2\relax
4501     \gdef\language#1}%
4502   \def\bbl@elt##1##2##3##4{}}%
4503   \bbl@languages
4504 \endgroup
4505 \fi
4506 \closeinl
```

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

```
4507 \if/\the\toks@/\else
4508   \errhelp{language.dat loads no language, only synonyms}
4509   \errmessage{Orphan language synonym}
4510 \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.

```
4511 \let\bbl@line\undefined
4512 \let\process@line\undefined
4513 \let\process@synonym\undefined
4514 \let\process@language\undefined
4515 \let\bbl@get@enc\undefined
4516 \let\bbl@hyph@enc\undefined
4517 \let\bbl@tempa\undefined
4518 \let\bbl@hook@loadkernel\undefined
4519 \let\bbl@hook@everylanguage\undefined
4520 \let\bbl@hook@loadpatterns\undefined
4521 \let\bbl@hook@loadexceptions\undefined
4522 \patterns
```

Here the code for iniTeX ends.

8 Font handling with fontspec

Add the bidi handler just before luaoffload, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
4523 <<{*More package options}>> ≡
4524 \chardef\bbl@bidimode\z@
4525 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4526 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
```

```

4527 \DeclareOption{bidi=basic-r}{\chardef\bbbl@bidimode=102 }
4528 \DeclareOption{bidi=bidi}{\chardef\bbbl@bidimode=201 }
4529 \DeclareOption{bidi=bidi-r}{\chardef\bbbl@bidimode=202 }
4530 \DeclareOption{bidi=bidi-l}{\chardef\bbbl@bidimode=203 }
4531 <</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. `bbbl@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.

```

4532 <<{*Font selection}>> ≡
4533 \bbbl@trace{Font handling with fontspec}
4534 \ifx\ExplSyntaxOn\undefined\else
4535   \def\bbbl@fs@warn@nx#1#2{% \bbbl@tempfs is the original macro
4536     \in{, #1,}{, no-script, language-not-exist,}%
4537     \ifin\else\bbbl@tempfs@nx{#1}{#2}\fi}
4538   \def\bbbl@fs@warn@nxx#1#2#3{%
4539     \in{, #1,}{, no-script, language-not-exist,}%
4540     \ifin\else\bbbl@tempfs@nxx{#1}{#2}{#3}\fi}
4541   \def\bbbl@loadfontspec{%
4542     \let\bbbl@loadfontspec\relax
4543     \ifx\fontspec\undefined
4544       \usepackage{fontspec}%
4545     \fi}%
4546 \fi
4547 \@onlypreamble\babelfont
4548 \newcommand\babelfont[2][{}]{% 1=langs/scripts 2=fam
4549   \bbbl@foreach{#1}{%
4550     \expandafter\ifx\csname date##1\endcsname\relax
4551       \IfFileExists{babel-##1.tex}%
4552         {\babelprovide{##1}}%
4553       {}%
4554     \fi}%
4555   \edef\bbbl@tempa{#1}%
4556   \def\bbbl@tempb{#2}% Used by \bbbl@bblfont
4557   \bbbl@loadfontspec
4558   \EnableBabelHook{babel-fontspec}% Just calls \bbbl@switchfont
4559   \bbbl@bblfont}
4560 \newcommand\bbbl@bblfont[2][{}]{% 1=features 2=fontname, @font=rm|sf|tt
4561   \bbbl@ifunset{\bbbl@tempb family}%
4562   {\bbbl@providefam{\bbbl@tempb}}%
4563   {}%
4564   % For the default font, just in case:
4565   \bbbl@ifunset{\bbbl@lsys\@languagename}{\bbbl@provide@lsys{\@languagename}}{}%
4566   \expandafter\bbbl@ifblank\expandafter{\bbbl@tempa}%
4567   {\bbbl@csarg\edef{\bbbl@tempb dflt@}{<>{#1}{#2}}% save \bbbl@rmdflt@
4568   \bbbl@exp{%
4569     \let\<\bbbl@tempb dflt@\@languagename>\<\bbbl@tempb dflt@>%
4570     \\\bbbl@font@set\<\bbbl@tempb dflt@\@languagename>%
4571     \<\bbbl@tempb default>\<\bbbl@tempb family>}}%
4572   {\bbbl@foreach\bbbl@tempa{% ie \bbbl@rmdflt@lang / *scrt
4573     \bbbl@csarg\def{\bbbl@tempb dflt@##1}{<>{#1}{#2}}}}%

```

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

```

4574 \def\bbbl@providefam#1{%
4575   \bbbl@exp{%
4576     \\\newcommand\<#1default>{}% Just define it
4577     \\\bbbl@add@list\bbbl@font@fams{#1}%
4578     \\\DeclareRobustCommand\<#1family>{%
4579       \\\not@math@alphabet\<#1family>\relax
4580       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4581       \\\fontfamily\<#1default>%

```

```

4582 \<ifx>\\UseHooks\\@undefined\<else>\\UseHook{#1family}\<fi>%
4583 \\selectfont}%
4584 \\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.

```

4585 \def\babelnostdfont#1{%
4586 \babel@ifunset{\babel@WFF@f@family}%
4587 {\babel@csarg\gdef{WFF@f@family}{}}% Flag, to avoid dupl warns
4588 \babel@infowarn{The current font is not a babel standard family:\\%
4589 #1%
4590 \fontname\font\\%
4591 There is nothing intrinsically wrong with this warning, and\\%
4592 you can ignore it altogether if you do not need these\\%
4593 families. But if they are used in the document, you should be\\%
4594 aware 'babel' will not set Script and Language for them, so\\%
4595 you may consider defining a new family with \string\babelfont.\\%
4596 See the manual for further details about \string\babelfont.\\%
4597 Reported}}
4598 {}}%
4599 \gdef\babel@switchfont{%
4600 \babel@ifunset{\babel@lsys@\language name}{\babel@provide@lsys{\language name}}}%
4601 \babel@exp{% eg Arabic -> arabic
4602 \lowercase{\edef\\babel@tempa{\babel@cl{sname}}}}%
4603 \babel@foreach\babel@font@fams{%
4604 \babel@ifunset{\babel@##1dflt@\language name}% (1) language?
4605 {\babel@ifunset{\babel@##1dflt@*\babel@tempa}% (2) from script?
4606 {\babel@ifunset{\babel@##1dflt@}% 2=F - (3) from generic?
4607 {}% 123=F - nothing!
4608 {\babel@exp{% 3=T - from generic
4609 \global\let\<\babel@##1dflt@\language name>%
4610 \<\babel@##1dflt@>}}}%
4611 {\babel@exp{% 2=T - from script
4612 \global\let\<\babel@##1dflt@\language name>%
4613 \<\babel@##1dflt@*\babel@tempa>}}}%
4614 {}% 1=T - language, already defined
4615 \def\babel@tempa{\babel@nostdfont{}}% TODO. Don't use \babel@tempa
4616 \babel@foreach\babel@font@fams{% don't gather with prev for
4617 \babel@ifunset{\babel@##1dflt@\language name}%
4618 {\babel@cs{famrst@##1}%
4619 \global\babel@csarg\let{famrst@##1}\relax}%
4620 {\babel@exp{% order is relevant. TODO: but sometimes wrong!
4621 \\babel@add\\originalTeX{%
4622 \\babel@font@rst{\babel@cl{##1dflt}}}%
4623 \<##1default>\<##1family>{##1}}}%
4624 \\babel@font@set\<\babel@##1dflt@\language name>% the main part!
4625 \<##1default>\<##1family>}}}%
4626 \babel@ifrestoring{ }\babel@tempa}}%

```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```

4627 \ifx\@family\@undefined\else % if latex
4628 \ifcase\babel@engine % if pdftex
4629 \let\babel@cckstdfont\relax
4630 \else
4631 \def\babel@cckstdfont{%
4632 \begingroup
4633 \global\let\babel@cckstdfont\relax
4634 \let\babel@tempa\@empty
4635 \babel@foreach\babel@font@fams{%
4636 \babel@ifunset{\babel@##1dflt@}%
4637 {\@nameuse{##1family}}%
4638 \babel@csarg\gdef{WFF@f@family}{}}% Flag
4639 \babel@exp{\\babel@add\\babel@tempa{* \<##1family>= \f@family\\}%

```

```

4640         \space\space\fontname\font\\\\}%
4641         \bbl@csarg\xdef{##ldflt@}{\f@family}%
4642         \expandafter\xdef\csname ##lddefault\endcsname{\f@family}%
4643     }%
4644     \ifx\bbl@tempa\empty\else
4645         \bbl@infowarn{The following font families will use the default\\%
4646             settings for all or some languages:\\%
4647         \bbl@tempa
4648         There is nothing intrinsically wrong with it, but\\%
4649         'babel' will no set Script and Language, which could\\%
4650         be relevant in some languages. If your document uses\\%
4651         these families, consider redefining them with \string\babelfont.\\%
4652         Reported}%
4653     \fi
4654 \endgroup}
4655 \fi
4656 \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, \TeX 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*`).

```

4657 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4658     \bbl@xin@{<>}{#1}%
4659     \ifin@
4660         \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4661     \fi
4662     \bbl@exp{%
4663         \def\\#2#1% eg, \rmdefault{\bbl@rmdflt@lang}
4664         \\bbl@ifsamestring{#2}{\f@family}%
4665         {\\#3%
4666             \\bbl@ifsamestring{\f@series}{\bfdefault}{\\bfseries}{}%
4667             \let\\bbl@tempa\relax}%
4668         {}%
4669 %     TODO - next should be global?, but even local does its job. I'm
4670 %     still not sure -- must investigate:
4671 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4672     \let\bbl@tempa\bbl@mapselect
4673     \edef\bbl@tempb{\bbl@stripslash#4}% Catcodes hack (better pass it).
4674     \bbl@exp{\\bbl@replace\\bbl@tempb{\bbl@stripslash\family/}{}}%
4675     \let\bbl@mapselect\relax
4676     \let\bbl@tempa\fam#4% eg, '\rmfamily', to be restored below
4677     \let#4\empty % Make sure \renewfontfamily is valid
4678     \bbl@exp{%
4679         \let\\bbl@tempa\pfa\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4680         \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4681         {\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4682         \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4683         {\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4684         \let\\bbl@tempfs@nx\<__fontspec_warning:nx>%
4685         \let\<__fontspec_warning:nx>\\bbl@fs@warn@nx
4686         \let\\bbl@tempfs@nxx\<__fontspec_warning:nxx>%
4687         \let\<__fontspec_warning:nxx>\\bbl@fs@warn@nxx
4688         \\renewfontfamily\\#4%
4689         [\bbl@cl{lsys},%
4690         \ifcase\bbl@engine\or RawFeature={family=\bbl@tempb},\fi
4691         #2]}{#3}% ie \bbl@exp{..}{#3}

```

```

4692 \bbl@exp{%
4693   \let<__fontspec_warning:nx>\\bbl@tempfs@nx
4694   \let<__fontspec_warning:nxx>\\bbl@tempfs@nxx}%
4695 \begingroup
4696   #4%
4697   \xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4698 \endgroup % TODO. Find better tests:
4699 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4700   {\expandafter\meaning\csname TU/#1/bx/sc\endcsname}%
4701 \ifin@
4702   \global\bbl@ccarg\let{TU/#1/bx/sc}{TU/#1/b/sc}%
4703 \fi
4704 \bbl@xin@{\string>\string s\string s\string u\string b\string*}%
4705   {\expandafter\meaning\csname TU/#1/bx/scit\endcsname}%
4706 \ifin@
4707   \global\bbl@ccarg\let{TU/#1/bx/scit}{TU/#1/b/scit}%
4708 \fi
4709 \let#4\bbl@temp@fam
4710 \bbl@exp{\let<\bbl@stripslash#4\space>}\bbl@temp@pfam
4711 \let\bbl@mapselect\bbl@tempe}%

```

font@rst and famrst are only used when there is no global settings, to save and restore de previous families. Not really necessary, but done for optimization.

```

4712 \def\bbl@font@rst#1#2#3#4{%
4713   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}

```

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

```

4714 \def\bbl@font@fams{rm,sf,tt}
4715 <</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.

```

4716 <<{*Footnote changes}>> ≡
4717 \bbl@trace{Bidi footnotes}
4718 \ifnum\bbl@bidimode>\z@ % Any bidi=
4719 \def\bbl@footnote#1#2#3{%
4720   \@ifnextchar[%
4721     {\bbl@footnote@o{#1}{#2}{#3}}%
4722     {\bbl@footnote@x{#1}{#2}{#3}}}
4723 \long\def\bbl@footnote@x#1#2#3#4{%
4724   \bgroup
4725   \select@language@x{\bbl@main@language}%
4726   \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4727   \egroup}
4728 \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4729   \bgroup
4730   \select@language@x{\bbl@main@language}%
4731   \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4732   \egroup}
4733 \def\bbl@footnotetext#1#2#3{%
4734   \@ifnextchar[%
4735     {\bbl@footnotetext@o{#1}{#2}{#3}}%
4736     {\bbl@footnotetext@x{#1}{#2}{#3}}}
4737 \long\def\bbl@footnotetext@x#1#2#3#4{%
4738   \bgroup
4739   \select@language@x{\bbl@main@language}%
4740   \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4741   \egroup}

```

```

4742 \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4743   \bgroup
4744     \select@language@x{\bbl@main@language}%
4745     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4746   \egroup}
4747 \def\BabelFootnote#1#2#3#4{%
4748   \ifx\bbl@fn@footnote\@undefined
4749     \let\bbl@fn@footnote\footnote
4750   \fi
4751   \ifx\bbl@fn@footnotetext\@undefined
4752     \let\bbl@fn@footnotetext\footnotetext
4753   \fi
4754   \bbl@ifblank{#2}%
4755     {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4756     \@namedef{\bbl@stripslash#1text}%
4757     {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4758     {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4759     \@namedef{\bbl@stripslash#1text}%
4760     {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4761 \fi
4762 <</Footnote changes>>

```

Now, the code.

```

4763 (*xetex)
4764 \def\BabelStringsDefault{unicode}
4765 \let\xebbl@stop\relax
4766 \AddBabelHook{xetex}{encodedcommands}{%
4767   \def\bbl@tempa{#1}%
4768   \ifx\bbl@tempa\@empty
4769     \XeTeXinputencoding"bytes"%
4770   \else
4771     \XeTeXinputencoding"#1"%
4772   \fi
4773   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4774 \AddBabelHook{xetex}{stopcommands}{%
4775   \xebbl@stop
4776   \let\xebbl@stop\relax}
4777 \def\bbl@intraspace#1 #2 #3\@@{%
4778   \bbl@csarg\gdef{xeisp@\languagename}%
4779   {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4780 \def\bbl@intrapenalty#1\@@{%
4781   \bbl@csarg\gdef{xeipn@\languagename}%
4782   {\XeTeXlinebreakpenalty #1\relax}}
4783 \def\bbl@provide@intraspace{%
4784   \bbl@xin@{/s}{/\bbl@cl{\lnbrk}}}%
4785   \ifin@ \else \bbl@xin@{/c}{/\bbl@cl{\lnbrk}} \fi
4786   \ifin@
4787     \bbl@ifunset{\bbl@intsp@\languagename}{}%
4788     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4789       \ifx\bbl@KVP@intraspace\@nnil
4790         \bbl@exp{%
4791           \bbl@intraspace\bbl@cl{intsp}\@@}%
4792       \fi
4793       \ifx\bbl@KVP@intrapenalty\@nnil
4794         \bbl@intrapenalty0\@@
4795       \fi
4796     \fi
4797     \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4798       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4799     \fi
4800     \ifx\bbl@KVP@intrapenalty\@nnil\else
4801       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4802     \fi

```

```

4803 \bbl@exp{%
4804 % TODO. Execute only once (but redundant):
4805 \\\bbl@add\<extras\language>{%
4806 \XeTeXlinebreaklocale "\bbl@cl{tbc}"%
4807 \<bbl@xeisp@\language>%
4808 \<bbl@xeipn@\language>}%
4809 \\\bbl@tglobal\<extras\language>%
4810 \\\bbl@add\<noextras\language>{%
4811 \XeTeXlinebreaklocale ""}%
4812 \\\bbl@tglobal\<noextras\language>}%
4813 \ifx\bbl@ispace\undefined
4814 \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4815 \ifx\AtBeginDocument\@notprerr
4816 \expandafter\@secondoftwo % to execute right now
4817 \fi
4818 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4819 \fi}%
4820 \fi}
4821 \ifx\DisableBabelHook\undefined\endinput\fi
4822 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4823 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4824 \DisableBabelHook{babel-fontspec}
4825 <<Font selection>>
4826 \def\bbl@provide@extra#1{}

```

10 Support for interchar

WIP.

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

```

4827 \ifnum\Xe@alloc@intercharclass<\thr@@
4828 \Xe@alloc@intercharclass\thr@@
4829 \fi
4830 \chardef\bbl@xe@class@default@=\z@
4831 \chardef\bbl@xe@class@cjkideograms@=\@ne
4832 \chardef\bbl@xe@class@cjkleftpunctuation@=\tw@
4833 \chardef\bbl@xe@class@cjkrightpunctuation@=\thr@@
4834 \chardef\bbl@xe@class@boundary@=4095
4835 \chardef\bbl@xe@class@ignore@=4096

```

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

```

4836 \AddBabelHook{babel-interchar}{beforeextras}{%
4837 \@nameuse{\bbl@xechars@\language}}
4838 \DisableBabelHook{babel-interchar}
4839 \protected\def\bbl@charclass#1{%
4840 \ifnum\count@<\z@
4841 \count@-\count@
4842 \loop
4843 \bbl@exp{%
4844 \\\babel@savevariable{\XeTeXcharclass`\Uchar\count@}}%
4845 \XeTeXcharclass\count@ \bbl@tempc
4846 \ifnum\count@<`#1\relax
4847 \advance\count@\@ne
4848 \repeat
4849 \else
4850 \babel@savevariable{\XeTeXcharclass`#1}%
4851 \XeTeXcharclass`#1 \bbl@tempc
4852 \fi
4853 \count@`#1\relax}

```


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

```

4854 \newcommand\babelcharclass[3]{%
4855   \EnableBabelHook{babel-interchar}%
4856   \bbl@csarg\newXeTeXintercharclass{xeclass@#2@#1}%
4857   \def\bbl@tempb##1{%
4858     \ifx##1@empty\else
4859       \ifx##1-
4860         \bbl@upto
4861       \else
4862         \bbl@charclass{%
4863           \ifcat\noexpand##1\relax\bbl@stripslash##1\else\string##1\fi}%
4864         \fi
4865         \expandafter\bbl@tempb
4866       \fi}%
4867   \bbl@ifunset{bbl@xechars@#1}%
4868   {\toks@{%
4869     \babel@savevariable\XeTeXinterchartokenstate
4870     \XeTeXinterchartokenstate\@ne
4871   }}%
4872   {\toks@\expandafter\expandafter\expandafter{%
4873     \csname bbl@xechars@#1\endcsname}}%
4874   \bbl@csarg\edef{xchars@#1}{%
4875     \the\toks@
4876     \bbl@usingxeclass\csname bbl@xeclass@#2@#1\endcsname
4877     \bbl@tempb#3@empty}}
4878 \protected\def\bbl@usingxeclass#1{\count@ \z@ \let\bbl@tempc#1}
4879 \protected\def\bbl@upto{%
4880   \ifnum\count@>\z@
4881     \advance\count@\@ne
4882     \count@-\count@
4883   \else\ifnum\count@=\z@
4884     \bbl@charclass{-}%
4885   \else
4886     \bbl@error{Double hyphens aren't allowed in \string\babelcharclass\\
4887               because it's potentially ambiguous}%
4888     {See the manual for further info}%
4889   \fi\fi}

```

And finally, the command with the code to be inserted. If the language doesn't define a class, then use the global one, as defined above. For the definition there is a intermediate macro, which can be 'disabled' with \bbl@ic@<label>@<lang>.

```

4890 \newcommand\babelinterchar[5][[]]{%
4891   \let\bbl@kv@label\@empty
4892   \bbl@forkv{#1}{\bbl@csarg\edef{kv@##1}{##2}}%
4893   \@namedef{\zap@space bbl@xeinter@\bbl@kv@label @#3@#4@#2 \@empty}%
4894   {\ifnum\language=\l@nohyphenation\else#5\fi}%
4895   \bbl@csarg\let{ic@\bbl@kv@label @#2}\@firstofone
4896   \bbl@exp{\bbl@for\bbl@tempa{\zap@space#3 \@empty}}{%
4897     \bbl@exp{\bbl@for\bbl@tempb{\zap@space#4 \@empty}}{%
4898       \XeTeXinterchartoks
4899       \@nameuse{bbl@xeclass@\bbl@tempa @%
4900         \bbl@ifunset{bbl@xeclass@\bbl@tempa @#2}{}}{#2}}
4901       \@nameuse{bbl@xeclass@\bbl@tempb @%
4902         \bbl@ifunset{bbl@xeclass@\bbl@tempb @#2}{}}{#2}}
4903     = \expandafter{%
4904       \csname bbl@ic@\bbl@kv@label @#2\expandafter\endcsname
4905       \csname\zap@space bbl@xeinter@\bbl@kv@label
4906         @#3@#4@#2 \@empty\endcsname}}}}

```

```

4907 \DeclareRobustCommand\enablelocaleinterchar[1]{%
4908   \bbl@ifunset{\bbl@ic@#1@\language}\language}%
4909   {\bbl@error
4910     {'#1' for '\language' cannot be enabled.\\%
4911       Maybe there is a typo.}%
4912     {See the manual for further details.}}%
4913   {\bbl@csarg\let{\ic@#1@\language}\@firstofone}}
4914 \DeclareRobustCommand\disablelocaleinterchar[1]{%
4915   \bbl@ifunset{\bbl@ic@#1@\language}\language}%
4916   {\bbl@error
4917     {'#1' for '\language' cannot be disabled.\\%
4918       Maybe there is a typo.}%
4919     {See the manual for further details.}}%
4920   {\bbl@csarg\let{\ic@#1@\language}\@gobble}}
4921 \xetex

```

10.1 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 \TeX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

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

```

4922 (*xetex | texxet)
4923 \providecommand\bbl@provide@intraspace{}
4924 \bbl@trace{Redefinitions for bidi layout}
4925 \def\bbl@sspre@caption{%
4926   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4927 \ifx\bbl@opt@layout@\nnil\else % if layout=..
4928 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4929 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4930 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4931   \def\@hangfrom#1{%
4932     \setbox\@tempboxa\hbox{#1}%
4933     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4934     \noindent\box\@tempboxa}
4935 \def\raggedright{%
4936   \let\\\@centercr
4937   \bbl@startskip\z@skip
4938   \@rightskip\@flushglue
4939   \bbl@endskip\@rightskip
4940   \parindent\z@
4941   \parfillskip\bbl@startskip}
4942 \def\raggedleft{%
4943   \let\\\@centercr
4944   \bbl@startskip\@flushglue
4945   \bbl@endskip\z@skip
4946   \parindent\z@
4947   \parfillskip\bbl@endskip}
4948 \fi
4949 \IfBabelLayout{lists}
4950 {\bbl@sreplace\list
4951   {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4952   \def\bbl@listleftmargin{%
4953     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4954   \ifcase\bbl@engine
4955     \def\labelenumii{}\theenumii}% pdfTeX doesn't reverse ()
4956     \def\p@enumiii{\p@enumii}\theenumii}%
4957   \fi
4958   \bbl@sreplace\@verbatim
4959   {\leftskip\@totalleftmargin}%
4960   {\bbl@startskip\textwidth

```

```

4961      \advance\bbbl@startskip-\linewidth}%
4962      \bbbl@sreplace\@verbatim
4963      {\rightskip\z@skip}%
4964      {\bbbl@endskip\z@skip}}%
4965      {}
4966 \IfBabelLayout{contents}
4967 {\bbbl@sreplace\@dottedtocline{\leftskip}{\bbbl@startskip}%
4968  \bbbl@sreplace\@dottedtocline{\rightskip}{\bbbl@endskip}}
4969 {}
4970 \IfBabelLayout{columns}
4971 {\bbbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbbl@outputbox}%
4972  \def\bbbl@outputbox#1{%
4973    \hb@xt@\textwidth{%
4974      \hskip\columnwidth
4975      \hfil
4976      {\normalcolor\vrule \@width\columnseprule}%
4977      \hfil
4978      \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4979      \hskip-\textwidth
4980      \hb@xt@\columnwidth{\box\@outputbox \hss}%
4981      \hskip\columnsep
4982      \hskip\columnwidth}}}%
4983 {}
4984 <<Footnote changes>>
4985 \IfBabelLayout{footnotes}%
4986 {\BabelFootnote\footnote\languagename{}}}%
4987 {\BabelFootnote\localfootnote\languagename{}}}%
4988 {\BabelFootnote\mainfootnote{}}}%
4989 {}

```

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.

```

4990 \IfBabelLayout{counters*}%
4991 {\bbbl@add\bbbl@opt@layout{.counters.}%
4992  \AddToHook{shipout/before}{%
4993    \let\bbbl@tempa\babelsublr
4994    \let\babelsublr\@firstofone
4995    \let\bbbl@save@thepage\thepage
4996    \protected@edef\thepage{\thepage}%
4997    \let\babelsublr\bbbl@tempa}%
4998  \AddToHook{shipout/after}{%
4999    \let\thepage\bbbl@save@thepage}}}%
5000 \IfBabelLayout{counters}%
5001 {\let\bbbl@latinarabic=\@arabic
5002  \def\@arabic#1{\babelsublr{\bbbl@latinarabic#1}}}%
5003  \let\bbbl@asciroman=\@roman
5004  \def\@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
5005  \let\bbbl@asciiRoman=\@Roman
5006  \def\@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}}}%
5007 \fi % end if layout
5008 </xetex | texpet)

```

10.2 8-bit TeX

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

```

5009 <*texpet>
5010 \def\bbbl@provide@extra#1{%
5011   % == auto-select encoding ==
5012   \ifx\bbbl@encoding@select\off\@empty\else
5013     \bbbl@ifunset{\bbbl@encoding@#1}%
5014     {\def\@elt##1{,##1,}%
5015      \edef\bbbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
5016      \count@\z@

```

```

5017 \bbl@foreach\bbl@tempe{%
5018 \def\bbl@tempd{##1}% Save last declared
5019 \advance\count@ \@ne}%
5020 \ifnum\count@>\@ne
5021 \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
5022 \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
5023 \bbl@replace\bbl@tempa{ },}%
5024 \global\bbl@csarg\let{encoding@#1}\@empty
5025 \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
5026 \ifin\@else % if main encoding included in ini, do nothing
5027 \let\bbl@tempb\relax
5028 \bbl@foreach\bbl@tempa{%
5029 \ifx\bbl@tempb\relax
5030 \bbl@xin@{,##1,}{,\bbl@tempe,}%
5031 \ifin\@def\bbl@tempb{##1}\fi
5032 \fi}%
5033 \ifx\bbl@tempb\relax\else
5034 \bbl@exp{%
5035 \global\<\bbl@add>\<\bbl@preextras@#1>\<\bbl@encoding@#1>}%
5036 \gdef\<\bbl@encoding@#1>{%
5037 \\\babel@save\\\f@encoding
5038 \\\bbl@add\\\originalTeX{\\\selectfont}%
5039 \\\fontencoding{\bbl@tempb}%
5040 \\\selectfont}}%
5041 \fi
5042 \fi
5043 \fi}%
5044 {}%
5045 \fi}
5046 \</texxet>

```

10.3 LuaTeX

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

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

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

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

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

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

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

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the base option); (2) at `hyphen.cfg`, to modify some macros; (3) in

the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```

5047 \*luatex>
5048 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
5049 \bbl@trace{Read language.dat}
5050 \ifx\bbl@readstream\undefined
5051   \csname newread\endcsname\bbl@readstream
5052 \fi
5053 \begingroup
5054   \toks@{}
5055   \count@ \z@ % 0=start, 1=0th, 2=normal
5056   \def\bbl@process@line#1#2 #3 #4 {%
5057     \ifx=#1%
5058       \bbl@process@synonym{#2}%
5059     \else
5060       \bbl@process@language{#1#2}{#3}{#4}%
5061     \fi
5062     \ignorespaces}
5063   \def\bbl@manylang{%
5064     \ifnum\bbl@last>\@ne
5065       \bbl@info{Non-standard hyphenation setup}%
5066     \fi
5067     \let\bbl@manylang\relax}
5068   \def\bbl@process@language#1#2#3{%
5069     \ifcase\count@
5070       \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
5071     \or
5072       \count@\tw@
5073     \fi
5074     \ifnum\count@=\tw@
5075       \expandafter\addlanguage\csname l@#1\endcsname
5076       \language\allocationnumber
5077       \chardef\bbl@last\allocationnumber
5078       \bbl@manylang
5079       \let\bbl@elt\relax
5080       \xdef\bbl@languages{%
5081         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
5082     \fi
5083     \the\toks@
5084     \toks@{}}
5085   \def\bbl@process@synonym@aux#1#2{%
5086     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
5087     \let\bbl@elt\relax
5088     \xdef\bbl@languages{%
5089       \bbl@languages\bbl@elt{#1}{#2}{}}}%
5090   \def\bbl@process@synonym#1{%
5091     \ifcase\count@
5092       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
5093     \or
5094       \ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}}%
5095     \else
5096       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
5097     \fi}
5098   \ifx\bbl@languages\undefined % Just a (sensible?) guess
5099     \chardef\l@english\z@
5100     \chardef\l@USenglish\z@
5101     \chardef\bbl@last\z@
5102     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
5103     \gdef\bbl@languages{%
5104       \bbl@elt{english}{0}{hyphen.tex}}%
5105     \bbl@elt{USenglish}{0}{}
5106   \else
5107     \global\let\bbl@languages@format\bbl@languages

```

```

5108 \def\bbl@elt#1#2#3#4{% Remove all except language 0
5109 \ifnum#2>\z@\else
5110 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
5111 \fi}%
5112 \xdef\bbl@languages{\bbl@languages}%
5113 \fi
5114 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}{}} % Define flags
5115 \bbl@languages
5116 \openin\bbl@readstream=language.dat
5117 \ifeof\bbl@readstream
5118 \bbl@warning{I couldn't find language.dat. No additional\\%
5119 patterns loaded. Reported}%
5120 \else
5121 \loop
5122 \endlinechar\m@ne
5123 \read\bbl@readstream to \bbl@line
5124 \endlinechar\^^M
5125 \if T\ifeof\bbl@readstream F\fi T\relax
5126 \ifx\bbl@line\empty\else
5127 \edef\bbl@line{\bbl@line\space\space\space}%
5128 \expandafter\bbl@process@line\bbl@line\relax
5129 \fi
5130 \repeat
5131 \fi
5132 \closein\bbl@readstream
5133 \endgroup
5134 \bbl@trace{Macros for reading patterns files}
5135 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
5136 \ifx\babelcatcodetablenum\undefined
5137 \ifx\newcatcodetable\undefined
5138 \def\babelcatcodetablenum{5211}
5139 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5140 \else
5141 \newcatcodetable\babelcatcodetablenum
5142 \newcatcodetable\bbl@pattcodes
5143 \fi
5144 \else
5145 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
5146 \fi
5147 \def\bbl@luapatterns#1#2{%
5148 \bbl@get@enc#1:.\@@@
5149 \setbox\z@\hbox\bgroup
5150 \begingroup
5151 \savecatcodetable\babelcatcodetablenum\relax
5152 \initcatcodetable\bbl@pattcodes\relax
5153 \catcodetable\bbl@pattcodes\relax
5154 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
5155 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~ =13
5156 \catcode`\@ =11 \catcode`\^^I=10 \catcode`\^^J=12
5157 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
5158 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
5159 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
5160 \input #1\relax
5161 \catcodetable\babelcatcodetablenum\relax
5162 \endgroup
5163 \def\bbl@tempa{#2}%
5164 \ifx\bbl@tempa\empty\else
5165 \input #2\relax
5166 \fi
5167 \egroup}%
5168 \def\bbl@patterns@lua#1{%
5169 \language=\expandafter\ifx\csname l@#1:f@encoding\endcsname\relax
5170 \csname l@#1\endcsname

```

```

5171 \edef\bbl@tempa{#1}%
5172 \else
5173 \csname l@#1:\f@encoding\endcsname
5174 \edef\bbl@tempa{#1:\f@encoding}%
5175 \fi\relax
5176 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
5177 \@ifundefined{bbl@hyphendata@the\language}%
5178 {\def\bbl@elt##1##2##3##4{%
5179 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5180 \def\bbl@tempb{##3}%
5181 \ifx\bbl@tempb\@empty\else % if not a synonymous
5182 \def\bbl@tempc{{##3}{##4}}%
5183 \fi
5184 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5185 \fi}%
5186 \bbl@languages
5187 \@ifundefined{bbl@hyphendata@the\language}%
5188 {\bbl@info{No hyphenation patterns were set for\%
5189 language '\bbl@tempa'. Reported}}%
5190 {\expandafter\expandafter\expandafter\bbl@luapatterns
5191 \csname bbl@hyphendata@the\language\endcsname}}}}
5192 \endinput\fi
5193 % Here ends \ifx\AddBabelHook\@undefined
5194 % A few lines are only read by hyphen.cfg
5195 \ifx\DisableBabelHook\@undefined
5196 \AddBabelHook{luatex}{everylanguage}{%
5197 \def\process@language##1##2##3{%
5198 \def\process@line####1####2 ####3 ####4 {}}}
5199 \AddBabelHook{luatex}{loadpatterns}{%
5200 \input #1\relax
5201 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
5202 {{#1}}}}
5203 \AddBabelHook{luatex}{loadexceptions}{%
5204 \input #1\relax
5205 \def\bbl@tempb##1##2{{##1}{#1}}%
5206 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
5207 {\expandafter\expandafter\expandafter\bbl@tempb
5208 \csname bbl@hyphendata@the\language\endcsname}}
5209 \endinput\fi
5210 % Here stops reading code for hyphen.cfg
5211 % The following is read the 2nd time it's loaded
5212 \begingroup % TODO - to a lua file
5213 \catcode`\%=12
5214 \catcode`\'=12
5215 \catcode`\%=12
5216 \catcode`\:=12
5217 \directlua{
5218 Babel = Babel or {}
5219 function Babel.bytes(line)
5220 return line:gsub("(.)",
5221 function (chr) return unicode.utf8.char(string.byte(chr)) end)
5222 end
5223 function Babel.begin_process_input()
5224 if luatexbase and luatexbase.add_to_callback then
5225 luatexbase.add_to_callback('process_input_buffer',
5226 Babel.bytes, 'Babel.bytes')
5227 else
5228 Babel.callback = callback.find('process_input_buffer')
5229 callback.register('process_input_buffer', Babel.bytes)
5230 end
5231 end
5232 function Babel.end_process_input ()
5233 if luatexbase and luatexbase.remove_from_callback then

```

```

5234     luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5235 else
5236     callback.register('process_input_buffer',Babel.callback)
5237 end
5238 end
5239 function Babel.addpatterns(pp, lg)
5240     local lg = lang.new(lg)
5241     local pats = lang.patterns(lg) or ''
5242     lang.clear_patterns(lg)
5243     for p in pp:gmatch('[^%s]+') do
5244         ss = ''
5245         for i in string.utfcharacters(p:gsub('%d', '')) do
5246             ss = ss .. '%d?' .. i
5247         end
5248         ss = ss:gsub('^%%d%?%.','%.') .. '%d?'
5249         ss = ss:gsub('%.%%d%?$', '%%.')
5250         pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5251         if n == 0 then
5252             tex.sprint(
5253                 [[\string\csname\space bbl@info\endcsname{New pattern: }]]
5254                 .. p .. [[{}]])
5255             pats = pats .. ' ' .. p
5256         else
5257             tex.sprint(
5258                 [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
5259                 .. p .. [[{}]])
5260         end
5261     end
5262     lang.patterns(lg, pats)
5263 end
5264 Babel.characters = Babel.characters or {}
5265 Babel.ranges = Babel.ranges or {}
5266 function Babel.hlist_has_bidi(head)
5267     local has_bidi = false
5268     local ranges = Babel.ranges
5269     for item in node.traverse(head) do
5270         if item.id == node.id'glyph' then
5271             local itemchar = item.char
5272             local chardata = Babel.characters[itemchar]
5273             local dir = chardata and chardata.d or nil
5274             if not dir then
5275                 for nn, et in ipairs(ranges) do
5276                     if itemchar < et[1] then
5277                         break
5278                     elseif itemchar <= et[2] then
5279                         dir = et[3]
5280                         break
5281                     end
5282                 end
5283             end
5284             if dir and (dir == 'al' or dir == 'r') then
5285                 has_bidi = true
5286             end
5287         end
5288     end
5289     return has_bidi
5290 end
5291 function Babel.set_chranges_b (script, chrng)
5292     if chrng == '' then return end
5293     texio.write('Replacing ' .. script .. ' script ranges')
5294     Babel.script_blocks[script] = {}
5295     for s, e in string.gmatch(chrng..' ', '(.)%.%.(-)%s') do
5296         table.insert(

```



```

5297         Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5298     end
5299 end
5300 function Babel.discard_sublr(str)
5301     if str:find( [[\string\indexentry]] ) and
5302        str:find( [[\string\babelsublr]] ) then
5303         str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5304                        function(m) return m:sub(2,-2) end )
5305     end
5306     return str
5307 end
5308 }
5309 \endgroup
5310 \ifx\newattribute\undefined\else % Test for plain
5311     \newattribute\bbl@attr@locale
5312     \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5313     \AddBabelHook{luatex}{beforeextras}{%
5314         \setattribute\bbl@attr@locale\localeid}
5315 \fi
5316 \def\BabelStringsDefault{unicode}
5317 \let\luabbl@stop\relax
5318 \AddBabelHook{luatex}{encodedcommands}{%
5319     \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5320     \ifx\bbl@tempa\bbl@tempb\else
5321         \directlua{Babel.begin_process_input()}%
5322         \def\luabbl@stop{%
5323             \directlua{Babel.end_process_input()}}%
5324     \fi}%
5325 \AddBabelHook{luatex}{stopcommands}{%
5326     \luabbl@stop
5327     \let\luabbl@stop\relax}
5328 \AddBabelHook{luatex}{patterns}{%
5329     \@ifundefined{bbl@hyphendata@the\language}%
5330     {\def\bbl@elt##1##2##3##4{%
5331         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5332         \def\bbl@tempb{##3}%
5333         \ifx\bbl@tempb\@empty\else % if not a synonymous
5334             \def\bbl@tempc{{##3}{##4}}}%
5335         \fi
5336         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5337     \fi}%
5338     \bbl@languages
5339     \@ifundefined{bbl@hyphendata@the\language}%
5340     {\bbl@info{No hyphenation patterns were set for\%
5341         language '#2'. Reported}}%
5342     {\expandafter\expandafter\expandafter\bbl@luapatterns
5343         \csname bbl@hyphendata@the\language\endcsname}}}%
5344 \@ifundefined{bbl@patterns@}{}%
5345 \begingroup
5346     \bbl@xin@{, \number\language, }{, \bbl@pttnlist}%
5347     \ifin@else
5348         \ifx\bbl@patterns@\@empty\else
5349             \directlua{ Babel.addpatterns(
5350                 [[\bbl@patterns@]], \number\language) }%
5351         \fi
5352         \@ifundefined{bbl@patterns@#1}%
5353         \@empty
5354         {\directlua{ Babel.addpatterns(
5355             [[\space\csname bbl@patterns@#1\endcsname]],
5356             \number\language) }}%
5357         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5358     \fi
5359 \endgroup}%

```

```

5360 \bbl@exp{%
5361   \bbl@ifunset{\bbl@prehc@{language}}{%
5362     {\bbl@ifblank{\bbl@cs{prehc@{language}}}{}}%
5363     {\prehyphenchar=\bbl@c{prehc}\relax}}}%

```

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

```

5364 \@onlypreamble\babelpatterns
5365 \AtEndOfPackage{%
5366   \newcommand\babelpatterns[2][\@empty]{%
5367     \ifx\bbl@patterns@relax
5368       \let\bbl@patterns@empty
5369     \fi
5370     \ifx\bbl@pttnlistemptyelse
5371       \bbl@warning{%
5372         You must not intermingle \string\selectlanguage\space and\%
5373         \string\babelpatterns\space or some patterns will not\%
5374         be taken into account. Reported}%
5375       \fi
5376       \ifxempty#1%
5377         \protected@edef\bbl@patterns@{\bbl@patterns@space#2}%
5378       \else
5379         \edef\bbl@tempb{\zap@space#1 \@empty}%
5380         \bbl@for\bbl@tempa\bbl@tempb{%
5381           \bbl@fixname\bbl@tempa
5382           \bbl@iflanguage\bbl@tempa{%
5383             \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5384               \ifundefined\bbl@patterns@\bbl@tempa}%
5385             \@empty
5386             {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5387             #2}}}%
5388       \fi}}

```

10.4 Southeast Asian scripts

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

```

5389% TODO - to a lua file
5390\directlua{
5391  Babel = Babel or {}
5392  Babel.linebreaking = Babel.linebreaking or {}
5393  Babel.linebreaking.before = {}
5394  Babel.linebreaking.after = {}
5395  Babel.locale = {} % Free to use, indexed by \localeid
5396  function Babel.linebreaking.add_before(func, pos)
5397    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5398    if pos == nil then
5399      table.insert(Babel.linebreaking.before, func)
5400    else
5401      table.insert(Babel.linebreaking.before, pos, func)
5402    end
5403  end
5404  function Babel.linebreaking.add_after(func)
5405    tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5406    table.insert(Babel.linebreaking.after, func)
5407  end
5408}
5409\def\bbl@intraspace#1 #2 #3\@{
5410  \directlua{

```

```

5411 Babel = Babel or {}
5412 Babel.intraspaces = Babel.intraspaces or {}
5413 Babel.intraspaces['\csname bbl@sbcpr@language\endcsname'] = %
5414 {b = #1, p = #2, m = #3}
5415 Babel.locale_props[\the\localeid].intraspace = %
5416 {b = #1, p = #2, m = #3}
5417 }}
5418 \def\bbl@intrapenalty#1\@@{%
5419 \directlua{
5420 Babel = Babel or {}
5421 Babel.intrapenalties = Babel.intrapenalties or {}
5422 Babel.intrapenalties['\csname bbl@sbcpr@language\endcsname'] = #1
5423 Babel.locale_props[\the\localeid].intrapenalty = #1
5424 }}
5425 \begingroup
5426 \catcode`\%=12
5427 \catcode`\^=14
5428 \catcode`\'=12
5429 \catcode`\~=12
5430 \gdef\bbl@seaintraspace{^
5431 \let\bbl@seaintraspace\relax
5432 \directlua{
5433 Babel = Babel or {}
5434 Babel.sea_enabled = true
5435 Babel.sea_ranges = Babel.sea_ranges or {}
5436 function Babel.set_chranges (script, chrng)
5437 local c = 0
5438 for s, e in string.gmatch(chrng..' ', '(.-%.(-)%s') do
5439 Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5440 c = c + 1
5441 end
5442 end
5443 function Babel.sea_disc_to_space (head)
5444 local sea_ranges = Babel.sea_ranges
5445 local last_char = nil
5446 local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5447 for item in node.traverse(head) do
5448 local i = item.id
5449 if i == node.id'glyph' then
5450 last_char = item
5451 elseif i == 7 and item.subtype == 3 and last_char
5452 and last_char.char > 0x0C99 then
5453 quad = font.getfont(last_char.font).size
5454 for lg, rg in pairs(sea_ranges) do
5455 if last_char.char > rg[1] and last_char.char < rg[2] then
5456 lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril
5457 local intraspace = Babel.intraspaces[lg]
5458 local intrapenalty = Babel.intrapenalties[lg]
5459 local n
5460 if intrapenalty ~= 0 then
5461 n = node.new(14, 0) ^% penalty
5462 n.penalty = intrapenalty
5463 node.insert_before(head, item, n)
5464 end
5465 n = node.new(12, 13) ^% (glue, spaceskip)
5466 node.setglue(n, intraspace.b * quad,
5467 intraspace.p * quad,
5468 intraspace.m * quad)
5469 node.insert_before(head, item, n)
5470 node.remove(head, item)
5471 end
5472 end
5473 end

```

```

5474     end
5475   end
5476 }^^
5477 \bbl@luahyphenate}

```

10.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.

```

5478 \catcode`\%=14
5479 \gdef\bbl@cjkintraspacespace{%
5480   \let\bbl@cjkintraspacespace\relax
5481   \directlua{
5482     Babel = Babel or {}
5483     require('babel-data-cjk.lua')
5484     Babel.cjk_enabled = true
5485     function Babel.cjk_linebreak(head)
5486       local GLYPH = node.id'glyph'
5487       local last_char = nil
5488       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5489       local last_class = nil
5490       local last_lang = nil
5491
5492       for item in node.traverse(head) do
5493         if item.id == GLYPH then
5494
5495           local lang = item.lang
5496
5497           local LOCALE = node.get_attribute(item,
5498             Babel.attr_locale)
5499           local props = Babel.locale_props[LOCALE]
5500
5501           local class = Babel.cjk_class[item.char].c
5502
5503           if props.cjk_quotes and props.cjk_quotes[item.char] then
5504             class = props.cjk_quotes[item.char]
5505           end
5506
5507           if class == 'cp' then class = 'cl' end % )) as CL
5508           if class == 'id' then class = 'I' end
5509
5510           local br = 0
5511           if class and last_class and Babel.cjk_breaks[last_class][class] then
5512             br = Babel.cjk_breaks[last_class][class]
5513           end
5514
5515           if br == 1 and props.linebreak == 'c' and
5516             lang ~= \the\l@nohyphenation\space and
5517             last_lang ~= \the\l@nohyphenation then
5518             local intrapenalty = props.intrapenalty
5519             if intrapenalty ~= 0 then
5520               local n = node.new(14, 0)      % penalty
5521               n.penalty = intrapenalty
5522               node.insert_before(head, item, n)
5523             end
5524             local intraspacespace = props.intraspacespace
5525             local n = node.new(12, 13)      % (glue, spaceskip)
5526             node.setglue(n, intraspacespace.b * quad,
5527               intraspacespace.p * quad,

```

```

5528             intraspace.m * quad)
5529         node.insert_before(head, item, n)
5530     end
5531
5532     if font.getfont(item.font) then
5533         quad = font.getfont(item.font).size
5534     end
5535     last_class = class
5536     last_lang = lang
5537     else % if penalty, glue or anything else
5538         last_class = nil
5539     end
5540 end
5541 lang.hyphenate(head)
5542 end
5543 }%
5544 \bbl@luahyphenate}
5545 \gdef\bbl@luahyphenate{%
5546 \let\bbl@luahyphenate\relax
5547 \directlua{
5548     luatexbase.add_to_callback('hyphenate',
5549     function (head, tail)
5550         if Babel.linebreaking.before then
5551             for k, func in ipairs(Babel.linebreaking.before) do
5552                 func(head)
5553             end
5554         end
5555         if Babel.cjk_enabled then
5556             Babel.cjk_linebreak(head)
5557         end
5558         lang.hyphenate(head)
5559         if Babel.linebreaking.after then
5560             for k, func in ipairs(Babel.linebreaking.after) do
5561                 func(head)
5562             end
5563         end
5564         if Babel.sea_enabled then
5565             Babel.sea_disc_to_space(head)
5566         end
5567     end,
5568     'Babel.hyphenate')
5569 }
5570 }
5571 \endgroup
5572 \def\bbl@provide@intraspace{%
5573 \bbl@ifunset{\bbl@intsp@{language}}{%
5574     {\expandafter\ifx\csname bbl@intsp@{language}\endcsname\@empty\else
5575         \bbl@xin@{c}{\bbl@cl{lnbrk}}}%
5576     \ifin@         % cjk
5577         \bbl@cjk@intraspace
5578         \directlua{
5579             Babel = Babel or {}
5580             Babel.locale_props = Babel.locale_props or {}
5581             Babel.locale_props[\the\localeid].linebreak = 'c'
5582         }%
5583         \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@@}%
5584         \ifx\bbl@KVP@intrapenalty\@nnil
5585             \bbl@intrapenalty0\@@
5586         \fi
5587     \else         % sea
5588         \bbl@sea@intraspace
5589         \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}\@@}%
5590         \directlua{

```

```

5591         Babel = Babel or {}
5592         Babel.sea_ranges = Babel.sea_ranges or {}
5593         Babel.set_chranges('\bbl@c{l{sbc}}',
5594                             '\bbl@c{l{chrng}}')
5595     }%
5596     \ifx\bbl@KVP@intrapenalty\@nnil
5597         \bbl@intrapenalty0@@
5598     \fi
5599 \fi
5600 \fi
5601 \ifx\bbl@KVP@intrapenalty\@nnil\else
5602     \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5603 \fi}}

```

10.6 Arabic justification

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

```

5604 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5605 \def\bblar@chars{%
5606     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5607     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5608     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5609 \def\bblar@elongated{%
5610     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5611     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5612     0649,064A}
5613 \begingroup
5614     \catcode\_ =11 \catcode\_:=11
5615     \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5616 \endgroup
5617 \gdef\bbl@arabicjust{% TODO. Allow for several locales.
5618     \let\bbl@arabicjust\relax
5619     \newattribute\bblar@kashida
5620     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5621     \bblar@kashida=\z@
5622     \bbl@patchfont{\bbl@parseja}{}%
5623     \directlua{
5624         Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5625         Babel.arabic.elong_map[\the\localeid] = {}
5626         luatexbase.add_to_callback('post_linebreak_filter',
5627             Babel.arabic.justify, 'Babel.arabic.justify')
5628         luatexbase.add_to_callback('hpack_filter',
5629             Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5630     }}%

```

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

```

5631 \def\bblar@fetchja#1#2#3#4{%
5632     \bbl@exp{\bbl@foreach{#1}}{%
5633         \bbl@ifunset\bblar@JE@##1{%
5634             {\setbox\z@\hbox{\textdir TRT ^^^200d\char"##1#2}}%
5635             {\setbox\z@\hbox{\textdir TRT ^^^200d\char"\@nameuse\bblar@JE@##1#2}}%
5636         \directlua{%
5637             local last = nil
5638             for item in node.traverse(tex.box[0].head) do
5639                 if item.id == node.id'glyph' and item.char > 0x600 and
5640                    not (item.char == 0x200D) then
5641                     last = item
5642                 end
5643             end
5644             Babel.arabic.#3['##1#4'] = last.char
5645         }}}

```

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?

```

5646 \gdef\bbl@parsejalt{%
5647   \ifx\addfontfeature\@undefined\else
5648     \bbl@xin@{/e}/{/\bbl@cl{\lnbrk}}%
5649     \ifin@
5650       \directlua{%
5651         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5652           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5653           tex.print([[string\cswname\space bbl@parsejalti\endcswname]])
5654         end
5655       }%
5656     \fi
5657   \fi}
5658 \gdef\bbl@parsejalti{%
5659   \begingroup
5660     \let\bbl@parsejalt\relax % To avoid infinite loop
5661     \edef\bbl@tempb{\fontid\font}%
5662     \bblar@nofswarn
5663     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5664     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5665     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5666     \addfontfeature{RawFeature+=jalt}%
5667     % \@namedef\bblar@JE@0643}{06AA}% todo: catch medial kaf
5668     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5669     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5670     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5671     \directlua{%
5672       for k, v in pairs(Babel.arabic.from) do
5673         if Babel.arabic.dest[k] and
5674           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5675           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5676             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5677         end
5678       end
5679     }%
5680   \endgroup}

```

The actual justification (inspired by CHICKENIZE).

```

5681 \begingroup
5682 \catcode`#=11
5683 \catcode`~=11
5684 \directlua{
5685
5686 Babel.arabic = Babel.arabic or {}
5687 Babel.arabic.from = {}
5688 Babel.arabic.dest = {}
5689 Babel.arabic.justify_factor = 0.95
5690 Babel.arabic.justify_enabled = true
5691 Babel.arabic.kashida_limit = -1
5692
5693 function Babel.arabic.justify(head)
5694   if not Babel.arabic.justify_enabled then return head end
5695   for line in node.traverse_id(node.id'hlist', head) do
5696     Babel.arabic.justify_hlist(head, line)
5697   end
5698   return head
5699 end
5700
5701 function Babel.arabic.justify_hbox(head, gc, size, pack)
5702   local has_inf = false
5703   if Babel.arabic.justify_enabled and pack == 'exactly' then
5704     for n in node.traverse_id(12, head) do

```

```

5705     if n.stretch_order > 0 then has_inf = true end
5706   end
5707   if not has_inf then
5708     Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5709   end
5710 end
5711 return head
5712 end
5713
5714 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5715   local d, new
5716   local k_list, k_item, pos_inline
5717   local width, width_new, full, k_curr, wt_pos, goal, shift
5718   local subst_done = false
5719   local elong_map = Babel.arabic.elong_map
5720   local cnt
5721   local last_line
5722   local GLYPH = node.id'glyph'
5723   local KASHIDA = Babel.attr_kashida
5724   local LOCALE = Babel.attr_locale
5725
5726   if line == nil then
5727     line = {}
5728     line.glue_sign = 1
5729     line.glue_order = 0
5730     line.head = head
5731     line.shift = 0
5732     line.width = size
5733   end
5734
5735   % Exclude last line. todo. But-- it discards one-word lines, too!
5736   % ? Look for glue = 12:15
5737   if (line.glue_sign == 1 and line.glue_order == 0) then
5738     elongs = {}      % Stores elongated candidates of each line
5739     k_list = {}      % And all letters with kashida
5740     pos_inline = 0   % Not yet used
5741
5742     for n in node.traverse_id(GLYPH, line.head) do
5743       pos_inline = pos_inline + 1 % To find where it is. Not used.
5744
5745       % Elongated glyphs
5746       if elong_map then
5747         local locale = node.get_attribute(n, LOCALE)
5748         if elong_map[locale] and elong_map[locale][n.font] and
5749           elong_map[locale][n.font][n.char] then
5750           table.insert(elongs, {node = n, locale = locale} )
5751           node.set_attribute(n.prev, KASHIDA, 0)
5752         end
5753       end
5754
5755       % Tatwil
5756       if Babel.kashida_wts then
5757         local k_wt = node.get_attribute(n, KASHIDA)
5758         if k_wt > 0 then % todo. parameter for multi inserts
5759           table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5760         end
5761       end
5762
5763     end % of node.traverse_id
5764
5765     if #elongs == 0 and #k_list == 0 then goto next_line end
5766     full = line.width
5767     shift = line.shift

```



```

5768 goal = full * Babel.arabic.justify_factor % A bit crude
5769 width = node.dimensions(line.head) % The 'natural' width
5770
5771 % == Elongated ==
5772 % Original idea taken from 'chickenize'
5773 while (#elongs > 0 and width < goal) do
5774     subst_done = true
5775     local x = #elongs
5776     local curr = elongs[x].node
5777     local oldchar = curr.char
5778     curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5779     width = node.dimensions(line.head) % Check if the line is too wide
5780     % Substitute back if the line would be too wide and break:
5781     if width > goal then
5782         curr.char = oldchar
5783         break
5784     end
5785     % If continue, pop the just substituted node from the list:
5786     table.remove(elongs, x)
5787 end
5788
5789 % == Tatwil ==
5790 if #k_list == 0 then goto next_line end
5791
5792 width = node.dimensions(line.head) % The 'natural' width
5793 k_curr = #k_list % Traverse backwards, from the end
5794 wt_pos = 1
5795
5796 while width < goal do
5797     subst_done = true
5798     k_item = k_list[k_curr].node
5799     if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5800         d = node.copy(k_item)
5801         d.char = 0x0640
5802         d.yoffset = 0 % TODO. From the prev char. But 0 seems safe.
5803         d.xoffset = 0
5804         line.head, new = node.insert_after(line.head, k_item, d)
5805         width_new = node.dimensions(line.head)
5806         if width > goal or width == width_new then
5807             node.remove(line.head, new) % Better compute before
5808             break
5809         end
5810         if Babel.fix_diacr then
5811             Babel.fix_diacr(k_item.next)
5812         end
5813         width = width_new
5814     end
5815     if k_curr == 1 then
5816         k_curr = #k_list
5817         wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5818     else
5819         k_curr = k_curr - 1
5820     end
5821 end
5822
5823 % Limit the number of tatweel by removing them. Not very efficient,
5824 % but it does the job in a quite predictable way.
5825 if Babel.arabic.kashida_limit > -1 then
5826     cnt = 0
5827     for n in node.traverse_id(GLYPH, line.head) do
5828         if n.char == 0x0640 then
5829             cnt = cnt + 1
5830             if cnt > Babel.arabic.kashida_limit then

```

```

5831         node.remove(line.head, n)
5832     end
5833     else
5834         cnt = 0
5835     end
5836 end
5837 end
5838
5839 ::next_line::
5840
5841 % Must take into account marks and ins, see luatex manual.
5842 % Have to be executed only if there are changes. Investigate
5843 % what's going on exactly.
5844 if subst_done and not gc then
5845     d = node.hpack(line.head, full, 'exactly')
5846     d.shift = shift
5847     node.insert_before(head, line, d)
5848     node.remove(head, line)
5849 end
5850 end % if process line
5851 end
5852 }
5853 \endgroup
5854 \fi\fi % ends Arabic just block: \ifnum\bbl@bidimode>100...

```

10.7 Common stuff

```

5855 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5856 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5857 \DisableBabelHook{babel-fontspec}
5858 <<Font selection>>

```

10.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a the function `Babel.locale_map`, which just traverse the node list to carry out the replacements. The table `loc_to_scr` stores the script range for each locale (whose id is the key), copied from this table (so that it can be modified on a locale basis); there is an intermediate table named `chr_to_loc` built on the fly for optimization, which maps a char to the locale. This locale is then used to get the `\language` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5859 % TODO - to a lua file
5860 \directlua{
5861 Babel.script_blocks = {
5862   ['dflt'] = {},
5863   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5864               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5865   ['Armn'] = {{0x0530, 0x058F}},
5866   ['Beng'] = {{0x0980, 0x09FF}},
5867   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5868   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5869   ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5870               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5871   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5872   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5873               {0xAB00, 0xAB2F}},
5874   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5875   % Don't follow strictly Unicode, which places some Coptic letters in
5876   % the 'Greek and Coptic' block
5877   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5878   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5879               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF}},

```

```

5880             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5881             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5882             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5883             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5884     ['Hebr'] = {{0x0590, 0x05FF}},
5885     ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5886             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5887     ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5888     ['Knda'] = {{0x0C80, 0x0CFF}},
5889     ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5890             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5891             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5892     ['Lao'] = {{0x0E80, 0x0EFF}},
5893     ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5894             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5895             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5896     ['Mahj'] = {{0x11150, 0x1117F}},
5897     ['Mlym'] = {{0x0D00, 0x0D7F}},
5898     ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5899     ['Orya'] = {{0x0B00, 0x0B7F}},
5900     ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5901     ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5902     ['Taml'] = {{0x0B80, 0x0BFF}},
5903     ['Telu'] = {{0x0C00, 0x0C7F}},
5904     ['Tfng'] = {{0x2D30, 0x2D7F}},
5905     ['Thai'] = {{0x0E00, 0x0E7F}},
5906     ['Tibt'] = {{0x0F00, 0x0FFF}},
5907     ['Vaii'] = {{0xA500, 0xA63F}},
5908     ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5909 }
5910
5911 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5912 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5913 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5914
5915 function Babel.locale_map(head)
5916     if not Babel.locale_mapped then return head end
5917
5918     local LOCALE = Babel.attr_locale
5919     local GLYPH = node.id('glyph')
5920     local inmath = false
5921     local toloc_save
5922     for item in node.traverse(head) do
5923         local toloc
5924         if not inmath and item.id == GLYPH then
5925             % Optimization: build a table with the chars found
5926             if Babel.chr_to_loc[item.char] then
5927                 toloc = Babel.chr_to_loc[item.char]
5928             else
5929                 for lc, maps in pairs(Babel.loc_to_scr) do
5930                     for _, rg in pairs(maps) do
5931                         if item.char >= rg[1] and item.char <= rg[2] then
5932                             Babel.chr_to_loc[item.char] = lc
5933                             toloc = lc
5934                             break
5935                         end
5936                     end
5937                 end
5938                 % Treat composite chars in a different fashion, because they
5939                 % 'inherit' the previous locale.
5940                 if (item.char >= 0x0300 and item.char <= 0x036F) or
5941                     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5942                     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then

```

```

5943         Babel.chr_to_loc[item.char] = -2000
5944         toloc = -2000
5945     end
5946     if not toloc then
5947         Babel.chr_to_loc[item.char] = -1000
5948     end
5949 end
5950 if toloc == -2000 then
5951     toloc = toloc_save
5952 elseif toloc == -1000 then
5953     toloc = nil
5954 end
5955 if toloc and Babel.locale_props[toloc] and
5956     Babel.locale_props[toloc].letters and
5957     tex.getcatcode(item.char) \string~= 11 then
5958     toloc = nil
5959 end
5960 if toloc and Babel.locale_props[toloc].script
5961     and Babel.locale_props[node.get_attribute(item, LOCALE)].script
5962     and Babel.locale_props[toloc].script ==
5963     Babel.locale_props[node.get_attribute(item, LOCALE)].script then
5964     toloc = nil
5965 end
5966 if toloc then
5967     if Babel.locale_props[toloc].lg then
5968         item.lang = Babel.locale_props[toloc].lg
5969         node.set_attribute(item, LOCALE, toloc)
5970     end
5971     if Babel.locale_props[toloc]['/'..item.font] then
5972         item.font = Babel.locale_props[toloc]['/'..item.font]
5973     end
5974 end
5975 toloc_save = toloc
5976 elseif not inmath and item.id == 7 then % Apply recursively
5977     item.replace = item.replace and Babel.locale_map(item.replace)
5978     item.pre      = item.pre and Babel.locale_map(item.pre)
5979     item.post     = item.post and Babel.locale_map(item.post)
5980 elseif item.id == node.id'math' then
5981     inmath = (item.subtype == 0)
5982 end
5983 end
5984 return head
5985 end
5986 }

```

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

```

5987 \newcommand\babelcharproperty[1]{%
5988   \count@=#1\relax
5989   \ifvmode
5990     \expandafter\bbl@chprop
5991   \else
5992     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5993       vertical mode (preamble or between paragraphs)}%
5994     {See the manual for further info}%
5995   \fi}
5996 \newcommand\bbl@chprop[3][\the\count@]{%
5997   \@tempcnta=#1\relax
5998   \bbl@ifunset{\bbl@chprop@#2}%
5999   {\bbl@error{No property named '#2'. Allowed values are\\%
6000     direction (bc), mirror (bmg), and linebreak (lb)}%
6001     {See the manual for further info}}%
6002   }%

```

```

6003 \loop
6004   \bbl@cs{chprop@#2}{#3}%
6005   \ifnum\count@<\@tempcnta
6006     \advance\count@\@ne
6007   \repeat}
6008 \def\bbl@chprop@direction#1{%
6009   \directlua{
6010     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6011     Babel.characters[\the\count@]['d'] = '#1'
6012   }}
6013 \let\bbl@chprop@bc\bbl@chprop@direction
6014 \def\bbl@chprop@mirror#1{%
6015   \directlua{
6016     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
6017     Babel.characters[\the\count@]['m'] = '\number#1'
6018   }}
6019 \let\bbl@chprop@bmg\bbl@chprop@mirror
6020 \def\bbl@chprop@linebreak#1{%
6021   \directlua{
6022     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
6023     Babel.cjk_characters[\the\count@]['c'] = '#1'
6024   }}
6025 \let\bbl@chprop@lb\bbl@chprop@linebreak
6026 \def\bbl@chprop@locale#1{%
6027   \directlua{
6028     Babel.chr_to_loc = Babel.chr_to_loc or {}
6029     Babel.chr_to_loc[\the\count@] =
6030       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
6031   }}

```

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.

```

6032 \directlua{
6033   Babel.nohyphenation = \the\l@nohyphenation
6034 }

```

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 $\text{function}(m) \text{ return } m[1]..m[1]..'-' \text{ end}$, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to $\text{function}(m) \text{ return } \text{Babel.capt_map}(m[1],1) \text{ end}$, 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).

```

6035 \begingroup
6036 \catcode`\~ = 12
6037 \catcode`\% = 12
6038 \catcode`\& = 14
6039 \catcode`\| = 12
6040 \gdef\babelprehyphenation{%&
6041   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}]
6042 \gdef\babelposthyphenation{%&
6043   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}]
6044 \gdef\bbl@settransform#1[#2]#3#4#5{%&
6045   \ifcase#1
6046     \bbl@activateprehyphen
6047   \or
6048     \bbl@activateposthyphen
6049   \fi
6050 \begingroup
6051   \def\babeltempa{\bbl@add@list\babeltempb}%&
6052   \let\babeltempb\@empty
6053   \def\bbl@tempa{#5}%&

```

```

6054 \bbl@replace\bbl@tempa{,}{,}%&% TODO. Ugly trick to preserve {}
6055 \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6056   \bbl@ifsamestring{##1}{remove}&%
6057   {\bbl@add@list\babeltempb{nil}}&%
6058   {\directlua{
6059     local rep = {[##1]=}
6060     rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6061     rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
6062     rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
6063     if #1 == 0 or #1 == 2 then
6064       rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6065         'space = { ' .. '%2, %3, %4' .. ' }')
6066       rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6067         'spacefactor = { ' .. '%2, %3, %4' .. ' }')
6068       rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
6069     else
6070       rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
6071       rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
6072       rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
6073     end
6074     tex.print([[string\babeltempa{}} .. rep .. [[]]])
6075   }}&%
6076 \bbl@foreach\babeltempb{&%
6077   \bbl@forkv{##1}{&%
6078     \in{,###1,},nil,step,data,remove,insert,string,no,pre,&%
6079     no,post,penalty,kashida,space,spacefactor,&%
6080     \ifin@else
6081     \bbl@error
6082     {Bad option '###1' in a transform.\\&%
6083     I'll ignore it but expect more errors}&%
6084     {See the manual for further info.}&%
6085     \fi}&%
6086 \let\bbl@kv@attribute\relax
6087 \let\bbl@kv@label\relax
6088 \let\bbl@kv@fonts\@empty
6089 \bbl@forkv{#2}{\bbl@csarg\edef{kv##1}{##2}}&%
6090 \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
6091 \ifx\bbl@kv@attribute\relax
6092   \ifx\bbl@kv@label\relax\else
6093     \bbl@exp{\bbl@trim\def\bbl@kv@fonts{\bbl@kv@fonts}}&%
6094     \bbl@replace\bbl@kv@fonts{ }{,}&%
6095     \edef\bbl@kv@attribute{\bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
6096     \count@ \z@
6097     \def\bbl@elt##1##2##3{&%
6098       \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
6099       {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
6100         {\count@ \@ne}&%
6101         {\bbl@error
6102           {Transforms cannot be re-assigned to different\\&%
6103           fonts. The conflict is in '\bbl@kv@label'.\\&%
6104           Apply the same fonts or use a different label}&%
6105           {See the manual for further details.}}}&%
6106         }}&%
6107     \bbl@transfont@list
6108     \ifnum\count@=\z@
6109       \bbl@exp{\global\bbl@add\bbl@transfont@list
6110         {\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
6111       \fi
6112     \bbl@ifunset{\bbl@kv@attribute}&%
6113     {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
6114     {}&%
6115     \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
6116   \fi

```

```

6117 \else
6118 \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
6119 \fi
6120 \directlua{
6121 local lbkr = Babel.linebreaking.replacements[#1]
6122 local u = unicode.utf8
6123 local id, attr, label
6124 if #1 == 0 then
6125 id = \the\csname bbl@id@@#3\endcsname\space
6126 else
6127 id = \the\csname l@#3\endcsname\space
6128 end
6129 \ifx\bbl@kv@attribute\relax
6130 attr = -1
6131 \else
6132 attr = luatexbase.registernumber'\bbl@kv@attribute'
6133 \fi
6134 \ifx\bbl@kv@label\relax\else &% Same refs:
6135 label = [[\bbl@kv@label]==]
6136 \fi
6137 &% Convert pattern:
6138 local patt = string.gsub([==[#4]==], '%s', '')
6139 if #1 == 0 then
6140 patt = string.gsub(patt, '|', ' ')
6141 end
6142 if not u.find(patt, '()', nil, true) then
6143 patt = '()' .. patt .. '()'
6144 end
6145 if #1 == 1 then
6146 patt = string.gsub(patt, '%(%)^', '^()')
6147 patt = string.gsub(patt, '%$(%)', '()$')
6148 end
6149 patt = u.gsub(patt, '{(.)}',
6150 function (n)
6151 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6152 end)
6153 patt = u.gsub(patt, '{(%x%x%x%x+)}',
6154 function (n)
6155 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
6156 end)
6157 lbkr[id] = lbkr[id] or {}
6158 table.insert(lbkr[id],
6159 { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
6160 }&%
6161 \endgroup}
6162 \endgroup
6163 \let\bbl@transfont@list\@empty
6164 \def\bbl@settransfont{%
6165 \global\let\bbl@settransfont\relax % Execute only once
6166 \gdef\bbl@transfont{%
6167 \def\bbl@elt####1####2####3{%
6168 \bbl@ifblank{####3}%
6169 {\count@tw@}% Do nothing if no fonts
6170 {\count@z@
6171 \bbl@vforeach{####3}{%
6172 \def\bbl@tempd{#####1}%
6173 \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
6174 \ifx\bbl@tempd\bbl@tempe
6175 \count@ne
6176 \else\ifx\bbl@tempd\bbl@transfam
6177 \count@ne
6178 \fi\fi}%
6179 \ifcase\count@

```

```

6180         \bbl@csarg\unsetattribute{ATR@###2@###1@###3}%
6181     \or
6182         \bbl@csarg\setattribute{ATR@###2@###1@###3}\@ne
6183     \fi}%
6184     \bbl@transfont@list}%
6185 \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
6186 \gdef\bbl@transfam{-unknown-}%
6187 \bbl@foreach\bbl@font@fams{%
6188     \AddToHook{##1family}{\def\bbl@transfam{##1}}%
6189     \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
6190     {\xdef\bbl@transfam{##1}}%
6191     {}}
6192 \DeclareRobustCommand\enablelocaletransform[1]{%
6193     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6194     {\bbl@error
6195         {'#1' for '\language' cannot be enabled.\\%
6196         Maybe there is a typo or it's a font-dependent transform}%
6197         {See the manual for further details.}}%
6198     {\bbl@csarg\setattribute{ATR@#1@\language @}\@ne}}
6199 \DeclareRobustCommand\disablelocaletransform[1]{%
6200     \bbl@ifunset{\bbl@ATR@#1@\language @}%
6201     {\bbl@error
6202         {'#1' for '\language' cannot be disabled.\\%
6203         Maybe there is a typo or it's a font-dependent transform}%
6204         {See the manual for further details.}}%
6205     {\bbl@csarg\unsetattribute{ATR@#1@\language @}}}
6206 \def\bbl@activateposthyphen{%
6207     \let\bbl@activateposthyphen\relax
6208     \directlua{
6209         require('babel-transforms.lua')
6210         Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6211     }}
6212 \def\bbl@activateprehyphen{%
6213     \let\bbl@activateprehyphen\relax
6214     \directlua{
6215         require('babel-transforms.lua')
6216         Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6217     }}

```

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.

```

6218 \newcommand\localeprehyphenation[1]{%
6219     \directlua{ Babel.string_prehyphenation([=[#1]=], \the\localeid) }}

```

10.9 Bidi

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

```

6220 \def\bbl@activate@preotf{%
6221     \let\bbl@activate@preotf\relax % only once
6222     \directlua{
6223         Babel = Babel or {}
6224         %
6225         function Babel.pre_otfload_v(head)
6226             if Babel.numbers and Babel.digits_mapped then
6227                 head = Babel.numbers(head)
6228             end
6229             if Babel.bidi_enabled then
6230                 head = Babel.bidi(head, false, dir)

```



```

6231     end
6232     return head
6233 end
6234 %
6235 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6236     if Babel.numbers and Babel.digits_mapped then
6237         head = Babel.numbers(head)
6238     end
6239     if Babel.bidi_enabled then
6240         head = Babel.bidi(head, false, dir)
6241     end
6242     return head
6243 end
6244 %
6245 luatexbase.add_to_callback('pre_linebreak_filter',
6246     Babel.pre_otfload_v,
6247     'Babel.pre_otfload_v',
6248     luatexbase.priority_in_callback('pre_linebreak_filter',
6249         'luaotfload.node_processor') or nil)
6250 %
6251 luatexbase.add_to_callback('hpack_filter',
6252     Babel.pre_otfload_h,
6253     'Babel.pre_otfload_h',
6254     luatexbase.priority_in_callback('hpack_filter',
6255         'luaotfload.node_processor') or nil)
6256 }}

```

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=`.

```

6257 \breakafterdirmode=1
6258 \ifnum\bbl@bidimode>\@ne % Any bidi= except default=1
6259     \let\bbl@beforeforeign\leavevmode
6260     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6261     \RequirePackage{luatexbase}
6262     \bbl@activate@preotf
6263     \directlua{
6264         require('babel-data-bidi.lua')
6265         \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6266             require('babel-bidi-basic.lua')
6267         \or
6268             require('babel-bidi-basic-r.lua')
6269         \fi}
6270     \newattribute\bbl@attr@dir
6271     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6272     \bbl@exp{\output{\bodydir\pagedir\the\output}}
6273 \fi
6274 \chardef\bbl@thetextdir\z@
6275 \chardef\bbl@thepardir\z@
6276 \def\bbl@getluadir#1{%
6277     \directlua{
6278         if tex.#ldir == 'TLT' then
6279             tex.sprint('0')
6280         elseif tex.#ldir == 'TRT' then
6281             tex.sprint('1')
6282         end}}
6283 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6284     \ifcase#3\relax
6285         \ifcase\bbl@getluadir{#1}\relax\else
6286             #2 TLT\relax
6287         \fi
6288     \else
6289         \ifcase\bbl@getluadir{#1}\relax

```

```

6290      #2 TRT\relax
6291      \fi
6292    \fi}
6293 % ..00PPTT, with masks 0xC (par dir) and 0x3 (text dir)
6294 \def\bbl@thedir{0}
6295 \def\bbl@textdir#1{%
6296   \bbl@setluadir{text}\textdir{#1}%
6297   \chardef\bbl@thetextdir#1\relax
6298   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6299   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6300 \def\bbl@pardir#1{% Used twice
6301   \bbl@setluadir{par}\pardir{#1}%
6302   \chardef\bbl@thepardir#1\relax}
6303 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}% Used once
6304 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}% Unused
6305 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once

```

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

```

6306 \ifnum\bbl@bidimode>\z@ % Any bidi=
6307   \def\bbl@insidemath{0}%
6308   \def\bbl@everymath{\def\bbl@insidemath{1}}
6309   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6310   \frozen@everymath\expandafter{%
6311     \expandafter\bbl@everymath\the\frozen@everymath}
6312   \frozen@everydisplay\expandafter{%
6313     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6314   \AtBeginDocument{
6315     \directlua{
6316       function Babel.math_box_dir(head)
6317         if not (token.get_macro('bbl@insidemath') == '0') then
6318           if Babel.hlist_has_bidi(head) then
6319             local d = node.new(node.id'dir')
6320             d.dir = '+TRT'
6321             node.insert_before(head, node.has_glyph(head), d)
6322             for item in node.traverse(head) do
6323               node.set_attribute(item,
6324                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6325             end
6326           end
6327         end
6328         return head
6329       end
6330       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6331         "Babel.math_box_dir", 0)
6332     } }%
6333 \fi

```

10.10 Layout

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

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

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

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

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

```

6334 \bbl@trace{Redefinitions for bidi layout}
6335 %
6336 <<{*More package options}>> ≡
6337 \chardef\bbl@eqnpos\z@
6338 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6339 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6340 <</More package options>>
6341 %
6342 \ifnum\bbl@bidimode>\z@ % Any bidi=
6343   \matheqdirmode\@ne % A luatex primitive
6344   \let\bbl@eqnodir\relax
6345   \def\bbl@eqdel{()}
6346   \def\bbl@eqnum{%
6347     {\normalfont\normalcolor
6348       \expandafter\@firstoftwo\bbl@eqdel
6349       \theequation
6350       \expandafter\@secondoftwo\bbl@eqdel}}
6351   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6352   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6353   \def\bbl@eqno@flip#1{%
6354     \ifdim\predisplaysize=-\maxdimen
6355       \eqno
6356       \hb@xt@.01pt{%
6357         \hb@xt@\displaywidth{\hss{#1}\glet\bbl@upset\@currentlabel}}\hss}%
6358     \else
6359       \leqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6360     \fi
6361     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6362   \def\bbl@leqno@flip#1{%
6363     \ifdim\predisplaysize=-\maxdimen
6364       \leqno
6365       \hb@xt@.01pt{%
6366         \hss\hb@xt@\displaywidth{\#1\glet\bbl@upset\@currentlabel}\hss}}%
6367     \else
6368       \eqno\hbox{#1}\glet\bbl@upset\@currentlabel}%
6369     \fi
6370     \bbl@exp{\def\\@currentlabel{\[bbl@upset]}}
6371   \AtBeginDocument{%
6372     \ifx\bbl@noamsmath\relax\else
6373       \ifx\maketag@@@undefined % Normal equation, eqnarray
6374         \AddToHook{env/equation/begin}{%
6375           \ifnum\bbl@thetextdir>\z@
6376             \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6377             \let\@eqnnum\bbl@eqnum
6378             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6379             \chardef\bbl@thetextdir\z@
6380             \bbl@add\normalfont{\bbl@eqnodir}%
6381             \ifcase\bbl@eqnpos
6382               \let\bbl@puteqno\bbl@eqno@flip
6383             \or
6384               \let\bbl@puteqno\bbl@leqno@flip
6385             \fi
6386             \fi}%
6387           \ifnum\bbl@eqnpos=\tw@\else
6388             \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6389             \fi

```

```

6390 \AddToHook{env/eqnarray/begin}{%
6391   \ifnum\bb1@thetextdir>\z@
6392     \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6393     \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6394     \chardef\bb1@thetextdir\z@
6395     \bb1@add\normalfont{\bb1@eqnodir}%
6396     \ifnum\bb1@eqnpos=\@ne
6397       \def\@eqnnum{%
6398         \setbox\z@\hbox{\bb1@eqnum}%
6399         \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6400     \else
6401       \let\@eqnnum\bb1@eqnum
6402     \fi
6403   \fi}
6404 % Hack. YA luatex bug?:
6405 \expandafter\bb1@sreplace\csname] \endcsname{${\eqno\kern.001pt$}}%
6406 \else % amstex
6407   \bb1@exp{% Hack to hide maybe undefined conditionals:
6408     \chardef\bb1@eqnpos=0%
6409     \<iftagsleft>1\<else>\<if\leqno>2\<fi>\<fi>\relax}%
6410   \ifnum\bb1@eqnpos=\@ne
6411     \let\bb1@ams@lap\hbox
6412   \else
6413     \let\bb1@ams@lap\llap
6414   \fi
6415   \ExplSyntaxOn % Required by \bb1@sreplace with \intertext@
6416   \bb1@sreplace\intertext@{\normalbaselines}%
6417   {\normalbaselines
6418     \ifx\bb1@eqnodir\relax\else\bb1@pardir\@ne\bb1@eqnodir\fi}%
6419   \ExplSyntaxOff
6420   \def\bb1@ams@tagbox#1#2{#1{\bb1@eqnodir#2}}% #1=hbox|@lap|flip
6421   \ifx\bb1@ams@lap\hbox % leqno
6422     \def\bb1@ams@flip#1{%
6423       \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6424   \else % eqno
6425     \def\bb1@ams@flip#1{%
6426       \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6427   \fi
6428   \def\bb1@ams@preset#1{%
6429     \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6430     \ifnum\bb1@thetextdir>\z@
6431       \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6432       \bb1@sreplace\textdef@{\hbox}{\bb1@ams@tagbox\hbox}%
6433       \bb1@sreplace\maketag@@@{\hbox}{\bb1@ams@tagbox#1}%
6434     \fi}%
6435   \ifnum\bb1@eqnpos=\tw@ \else
6436     \def\bb1@ams@equation{%
6437       \def\bb1@mathboxdir{\def\bb1@insidemath{1}}%
6438       \ifnum\bb1@thetextdir>\z@
6439         \edef\bb1@eqnodir{\noexpand\bb1@textdir{\the\bb1@thetextdir}}%
6440         \chardef\bb1@thetextdir\z@
6441         \bb1@add\normalfont{\bb1@eqnodir}%
6442         \ifcase\bb1@eqnpos
6443           \def\veqno##1##2{\bb1@eqno@flip{##1##2}}%
6444         \or
6445           \def\veqno##1##2{\bb1@leqno@flip{##1##2}}%
6446         \fi
6447       \fi}%
6448   \AddToHook{env/equation/begin}{\bb1@ams@equation}%
6449   \AddToHook{env/equation*/begin}{\bb1@ams@equation}%
6450 \fi
6451 \AddToHook{env/cases/begin}{\bb1@ams@preset\bb1@ams@lap}%
6452 \AddToHook{env/multline/begin}{\bb1@ams@preset\hbox}%

```

```

6453 \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6454 \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6455 \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6456 \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6457 \AddToHook{env/alignat/begin}{\bbl@ams@preset\bbl@ams@lap}%
6458 \AddToHook{env/alignat*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6459 \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6460 % Hackish, for proper alignment. Don't ask me why it works!:
6461 \bbl@exp{% Avoid a 'visible' conditional
6462   \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\tag*{}<fi>}%
6463   \\\AddToHook{env/alignat*/end}{\<iftag@>\<else>\\tag*{}<fi>}}%
6464 \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6465 \AddToHook{env/split/before}{%
6466   \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6467   \ifnum\bbl@thetextdir>\z@
6468     \bbl@ifsamestring@currentenv{equation}%
6469     {\ifx\bbl@ams@lap\hbox % leqno
6470       \def\bbl@ams@flip#1{%
6471         \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6472       \else
6473         \def\bbl@ams@flip#1{%
6474           \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}%
6475         \fi}%
6476       }%
6477     \fi}%
6478   \fi\fi}
6479 \fi
6480 \def\bbl@provide@extra#1{%
6481   % == Counters: mapdigits ==
6482   % Native digits
6483   \ifx\bbl@KVP@mapdigits\@nnil\else
6484     \bbl@ifunset{\bbl@dgnat@language}{%
6485       {\RequirePackage{luatexbase}%
6486        \bbl@activate@preotf
6487        \directlua{
6488          Babel = Babel or {} %% -> presets in luababel
6489          Babel.digits_mapped = true
6490          Babel.digits = Babel.digits or {}
6491          Babel.digits[\the\localeid] =
6492            table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6493          if not Babel.numbers then
6494            function Babel.numbers(head)
6495              local LOCALE = Babel.attr_locale
6496              local GLYPH = node.id'glyph'
6497              local inmath = false
6498              for item in node.traverse(head) do
6499                if not inmath and item.id == GLYPH then
6500                  local temp = node.get_attribute(item, LOCALE)
6501                  if Babel.digits[temp] then
6502                    local chr = item.char
6503                    if chr > 47 and chr < 58 then
6504                      item.char = Babel.digits[temp][chr-47]
6505                    end
6506                  end
6507                elseif item.id == node.id'math' then
6508                  inmath = (item.subtype == 0)
6509                end
6510              end
6511              return head
6512            end
6513          end
6514        }}%
6515      \fi

```

```

6516 % == transforms ==
6517 \ifx\bbI@KVP@transforms\@nnil\else
6518   \def\bbI@elt##1##2##3{%
6519     \in@{${transforms.}}{##1}%
6520     \ifin@
6521       \def\bbI@tempa{##1}%
6522       \bbI@replace\bbI@tempa{transforms.}{}%
6523       \bbI@carg\bbI@transforms{babel\bbI@tempa}{##2}{##3}%
6524     \fi}%
6525   \csname bbl@inidata@\language\endcsname
6526   \bbI@release@transforms\relax % \relax closes the last item.
6527 \fi}
6528 % Start tabular here:
6529 \def\localerestoredirs{%
6530   \ifcase\bbI@thetextdir
6531     \ifnum\textdirection=\z@\else\textdir TLT\fi
6532   \else
6533     \ifnum\textdirection=\@ne\else\textdir TRT\fi
6534   \fi
6535   \ifcase\bbI@thepardir
6536     \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6537   \else
6538     \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6539   \fi}
6540 \IfBabelLayout{tabular}%
6541   {\chardef\bbI@tabular@mode\tw@}% All RTL
6542   {\IfBabelLayout{notabular}%
6543     {\chardef\bbI@tabular@mode\z@}%
6544     {\chardef\bbI@tabular@mode\@ne}% Mixed, with LTR cols
6545   \ifnum\bbI@bidimode>\@ne % Any lua bidi= except default=1
6546     \ifcase\bbI@tabular@mode\or % 1
6547       \let\bbI@parabefore\relax
6548       \AddToHook{para/before}{\bbI@parabefore}
6549       \AtBeginDocument{%
6550         \bbI@replace\@tabular{$}{$}%
6551         \def\bbI@insidemath{0}%
6552         \def\bbI@parabefore{\localerestoredirs}}%
6553       \ifnum\bbI@tabular@mode=\@ne
6554         \bbI@ifunset{@tabclassz}{}%
6555         \bbI@exp{% Hide conditionals
6556           \\\bbI@sreplace\\ \@tabclassz
6557             {\<ifcase>\\ \@chnum}%
6558             {\\\localerestoredirs\<ifcase>\\ \@chnum}}}%
6559         \@ifpackageloaded{colortbl}%
6560           {\bbI@sreplace\@classz
6561             {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6562           {\@ifpackageloaded{array}%
6563             {\bbI@exp{% Hide conditionals
6564               \\\bbI@sreplace\\ \@classz
6565                 {\<ifcase>\\ \@chnum}%
6566                 {\bgroup\\ \localerestoredirs\<ifcase>\\ \@chnum}%
6567               \\\bbI@sreplace\\ \@classz
6568                 {\\\do@row@strut\<fi>}{\\do@row@strut\<fi>\egroup}}}%
6569             {}}%
6570       \fi}%
6571   \or % 2
6572     \let\bbI@parabefore\relax
6573     \AddToHook{para/before}{\bbI@parabefore}%
6574     \AtBeginDocument{%
6575       \@ifpackageloaded{colortbl}%
6576         {\bbI@replace\@tabular{$}{$}%
6577         \def\bbI@insidemath{0}%
6578         \def\bbI@parabefore{\localerestoredirs}}%

```

```

6579      \bbl@sreplace\@classz
6580      {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6581      {}}%
6582  \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.

```

6583  \AtBeginDocument{%
6584    \@ifpackageloaded{multicol}%
6585    {\toks@expandafter{\multi@column@out}%
6586     \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6587    {}%
6588    \@ifpackageloaded{paracol}%
6589    {\edef\pcol@output{%
6590     \bodydir\pagedir\unexpanded\expandafter{\pcol@output}}}%
6591    {}}%
6592  \fi
6593  \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout

```

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

```

6594  \ifnum\bbl@bidimode>\z@ % Any bidi=
6595  \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6596    \bbl@exp{%
6597      \def\\bbl@insidemath{0}%
6598      \mathdir\the\bodydir
6599      #1%          Once entered in math, set boxes to restore values
6600      \<ifmmode>%
6601        \everyvbox{%
6602          \the\everyvbox
6603          \bodydir\the\bodydir
6604          \mathdir\the\mathdir
6605          \everyhbox{\the\everyhbox}%
6606          \everyvbox{\the\everyvbox}}%
6607        \everyhbox{%
6608          \the\everyhbox
6609          \bodydir\the\bodydir
6610          \mathdir\the\mathdir
6611          \everyhbox{\the\everyhbox}%
6612          \everyvbox{\the\everyvbox}}%
6613        \<fi>}}%
6614  \def\@hangfrom#1{%
6615    \setbox\@tempboxa\hbox{#1}%
6616    \hangindent\wd\@tempboxa
6617    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6618      \shapemode\@ne
6619    \fi
6620    \noindent\box\@tempboxa}
6621  \fi
6622  \IfBabelLayout{tabular}
6623  {\let\bbl@OL@@tabular\@tabular
6624   \bbl@replace\@tabular{\$}\bbl@nextfake$}%
6625   \let\bbl@NL@@tabular\@tabular
6626   \AtBeginDocument{%
6627     \ifx\bbl@NL@@tabular\@tabular\else
6628       \bbl@exp{\\in@{\bbl@nextfake}\[@tabular]}}%
6629     \ifin@
6630       \bbl@replace\@tabular{\$}\bbl@nextfake$}%
6631     \fi
6632     \let\bbl@NL@@tabular\@tabular
6633   \fi}}

```

```

6634 {}
6635 \IfBabelLayout{lists}
6636 {\let\bbl@OL@list\list
6637 \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6638 \let\bbl@NL@list\list
6639 \def\bbl@listparshape#1#2#3{%
6640 \parshape #1 #2 #3 %
6641 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6642 \shapemode\tw@
6643 \fi}}
6644 {}
6645 \IfBabelLayout{graphics}
6646 {\let\bbl@pictresetdir\relax
6647 \def\bbl@pictsetdir#1{%
6648 \ifcase\bbl@thetextdir
6649 \let\bbl@pictresetdir\relax
6650 \else
6651 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6652 \or\textdir TLT
6653 \else\bodydir TLT \textdir TLT
6654 \fi
6655 % \(\text|par)dir required in pgf:
6656 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6657 \fi}%
6658 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6659 \directlua{
6660 Babel.get_picture_dir = true
6661 Babel.picture_has_bidi = 0
6662 %
6663 function Babel.picture_dir (head)
6664 if not Babel.get_picture_dir then return head end
6665 if Babel.hlist_has_bidi(head) then
6666 Babel.picture_has_bidi = 1
6667 end
6668 return head
6669 end
6670 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6671 "Babel.picture_dir")
6672 }%
6673 \AtBeginDocument{%
6674 \def\LS@rot{%
6675 \setbox\@outputbox\vbox{%
6676 \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}%
6677 \long\def\put(#1,#2)#3{%
6678 \@killglue
6679 % Try:
6680 \ifx\bbl@pictresetdir\relax
6681 \def\bbl@tempc{0}%
6682 \else
6683 \directlua{
6684 Babel.get_picture_dir = true
6685 Babel.picture_has_bidi = 0
6686 }%
6687 \setbox\z@\hb@xt@\z@{%
6688 \@defaultunitsset\@tempdimc{#1}\unitlength
6689 \kern\@tempdimc
6690 #3\hss}% TODO: #3 executed twice (below). That's bad.
6691 \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6692 \fi
6693 % Do:
6694 \@defaultunitsset\@tempdimc{#2}\unitlength
6695 \raise\@tempdimc\hb@xt@\z@{%
6696 \@defaultunitsset\@tempdimc{#1}\unitlength

```



```

6697 \kern\@tempdimc
6698 {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
6699 \ignorespaces}%
6700 \MakeRobust\put}%
6701 \AtBeginDocument
6702 {\AddToHook{cmd/diagbox@pict/before}{\let\bbbl@pictsetdir@gobble}%
6703 \ifx\pgfpicture@undefined\else % TODO. Allow deactivate?
6704 \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir@ne}%
6705 \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
6706 \bbbl@add\pgfsys@beginpicture{\bbbl@pictsetdir\z@}%
6707 \fi
6708 \ifx\tikzpicture@undefined\else
6709 \AddToHook{env/tikzpicture/begin}{\bbbl@pictsetdir\tw}%
6710 \bbbl@add\tikz@atbegin@node{\bbbl@pictresetdir}%
6711 \bbbl@sreplace\tikz{\begingroup}{\begingroup\bbbl@pictsetdir\tw}%
6712 \fi
6713 \ifx\tcolorbox@undefined\else
6714 \def\tcb@drawing@env@begin{%
6715 \csname tcb@before@tcb@split@state\endcsname
6716 \bbbl@pictsetdir\tw@
6717 \begin{\kvtcb@graphenv}%
6718 \tcb@bbdraw%
6719 \tcb@apply@graph@patches
6720 }%
6721 \def\tcb@drawing@env@end{%
6722 \end{\kvtcb@graphenv}%
6723 \bbbl@pictresetdir
6724 \csname tcb@after@tcb@split@state\endcsname
6725 }%
6726 \fi
6727 }}
6728 {}

```

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.

```

6729 \IfBabelLayout{counters*}%
6730 {\bbbl@add\bbbl@opt@layout{.counters.}%
6731 \directlua{
6732 \luatexbase.add_to_callback("process_output_buffer",
6733 Babel.discard_sublr , "Babel.discard_sublr") }%
6734 }}
6735 \IfBabelLayout{counters}%
6736 {\let\bbbl@0L@@textsuperscript@textsuperscript
6737 \bbbl@sreplace@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6738 \let\bbbl@latinarabic=@arabic
6739 \let\bbbl@0L@@arabic@arabic
6740 \def@arabic#1{\babelsublr{\bbbl@latinarabic#1}}%
6741 \@ifpackagewith{babel}{bidi=default}%
6742 {\let\bbbl@asciroman=@roman
6743 \let\bbbl@0L@@roman@roman
6744 \def@roman#1{\babelsublr{\ensureascii{\bbbl@asciroman#1}}}%
6745 \let\bbbl@asciiRoman=@Roman
6746 \let\bbbl@0L@@roman@Roman
6747 \def@Roman#1{\babelsublr{\ensureascii{\bbbl@asciiRoman#1}}}%
6748 \let\bbbl@0L@labelenumii\labelenumii
6749 \def\labelenumii{\theenumii}%
6750 \let\bbbl@0L@p@enumiii\p@enumiii
6751 \def\p@enumiii{\p@enumii}\theenumii}}{}{}
6752 <<Footnote changes>>
6753 \IfBabelLayout{footnotes}%
6754 {\let\bbbl@0L@footnote\footnote
6755 \BabelFootnote\footnote\languagename{}}}%

```

```

6756 \BabelFootnote\localfootnote\languagename{}{}%
6757 \BabelFootnote\mainfootnote{}{}{}
6758 {}

```

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

```

6759 \IfBabelLayout{extras}%
6760 {\bbl@ncarg\let\bbl@OL@underline{underline }%
6761 \bbl@carg\bbl@sreplace{underline }%
6762 {\$@@underline}{\bgroup\bbl@nextfake$@@underline}%
6763 \bbl@carg\bbl@sreplace{underline }%
6764 {\m@th$}\m@th$\egroup}%
6765 \let\bbl@OL@LaTeXe\LaTeXe
6766 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6767 \if b\expandafter\@car\@fseries\@nil\boldmath\fi
6768 \babelsublr{%
6769 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
6770 {}
6771 \luatex

```

10.11 Lua: transforms

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

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

```

6772 (*transforms)
6773 Babel.linebreaking.replacements = {}
6774 Babel.linebreaking.replacements[0] = {} -- pre
6775 Babel.linebreaking.replacements[1] = {} -- post
6776
6777 -- Discretionaries contain strings as nodes
6778 function Babel.str_to_nodes(fn, matches, base)
6779   local n, head, last
6780   if fn == nil then return nil end
6781   for s in string.utfvalues(fn(matches)) do
6782     if base.id == 7 then
6783       base = base.replace
6784     end
6785     n = node.copy(base)
6786     n.char = s
6787     if not head then
6788       head = n
6789     else
6790       last.next = n
6791     end
6792     last = n
6793   end
6794   return head
6795 end
6796
6797 Babel.fetch_subtext = {}
6798
6799 Babel.ignore_pre_char = function(node)
6800   return (node.lang == Babel.nohyphenation)
6801 end

```

```

6802
6803 -- Merging both functions doesn't seem feasible, because there are too
6804 -- many differences.
6805 Babel.fetch_subtext[0] = function(head)
6806   local word_string = ''
6807   local word_nodes = {}
6808   local lang
6809   local item = head
6810   local inmath = false
6811
6812   while item do
6813
6814     if item.id == 11 then
6815       inmath = (item.subtype == 0)
6816     end
6817
6818     if inmath then
6819       -- pass
6820
6821     elseif item.id == 29 then
6822       local locale = node.get_attribute(item, Babel.attr_locale)
6823
6824       if lang == locale or lang == nil then
6825         lang = lang or locale
6826         if Babel.ignore_pre_char(item) then
6827           word_string = word_string .. Babel.us_char
6828         else
6829           word_string = word_string .. unicode.utf8.char(item.char)
6830         end
6831         word_nodes[#word_nodes+1] = item
6832       else
6833         break
6834       end
6835
6836     elseif item.id == 12 and item.subtype == 13 then
6837       word_string = word_string .. ' '
6838       word_nodes[#word_nodes+1] = item
6839
6840     -- Ignore leading unrecognized nodes, too.
6841     elseif word_string ~= '' then
6842       word_string = word_string .. Babel.us_char
6843       word_nodes[#word_nodes+1] = item -- Will be ignored
6844     end
6845
6846     item = item.next
6847   end
6848
6849   -- Here and above we remove some trailing chars but not the
6850   -- corresponding nodes. But they aren't accessed.
6851   if word_string:sub(-1) == ' ' then
6852     word_string = word_string:sub(1,-2)
6853   end
6854   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6855   return word_string, word_nodes, item, lang
6856 end
6857
6858 Babel.fetch_subtext[1] = function(head)
6859   local word_string = ''
6860   local word_nodes = {}
6861   local lang
6862   local item = head
6863   local inmath = false
6864

```

```

6865 while item do
6866
6867     if item.id == 11 then
6868         inmath = (item.subtype == 0)
6869     end
6870
6871     if inmath then
6872         -- pass
6873
6874     elseif item.id == 29 then
6875         if item.lang == lang or lang == nil then
6876             if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6877                 lang = lang or item.lang
6878                 word_string = word_string .. unicode.utf8.char(item.char)
6879                 word_nodes[#word_nodes+1] = item
6880             end
6881         else
6882             break
6883         end
6884
6885     elseif item.id == 7 and item.subtype == 2 then
6886         word_string = word_string .. '='
6887         word_nodes[#word_nodes+1] = item
6888
6889     elseif item.id == 7 and item.subtype == 3 then
6890         word_string = word_string .. '|'
6891         word_nodes[#word_nodes+1] = item
6892
6893     -- (1) Go to next word if nothing was found, and (2) implicitly
6894     -- remove leading USs.
6895     elseif word_string == '' then
6896         -- pass
6897
6898     -- This is the responsible for splitting by words.
6899     elseif (item.id == 12 and item.subtype == 13) then
6900         break
6901
6902     else
6903         word_string = word_string .. Babel.us_char
6904         word_nodes[#word_nodes+1] = item -- Will be ignored
6905     end
6906
6907     item = item.next
6908 end
6909
6910 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6911 return word_string, word_nodes, item, lang
6912 end
6913
6914 function Babel.pre_hyphenate_replace(head)
6915     Babel.hyphenate_replace(head, 0)
6916 end
6917
6918 function Babel.post_hyphenate_replace(head)
6919     Babel.hyphenate_replace(head, 1)
6920 end
6921
6922 Babel.us_char = string.char(31)
6923
6924 function Babel.hyphenate_replace(head, mode)
6925     local u = unicode.utf8
6926     local lbkr = Babel.linebreaking.replacements[mode]
6927

```

```

6928 local word_head = head
6929
6930 while true do -- for each subtext block
6931
6932     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6933
6934     if Babel.debug then
6935         print()
6936         print((mode == 0) and '@@@<' or '@@@>', w)
6937     end
6938
6939     if nw == nil and w == '' then break end
6940
6941     if not lang then goto next end
6942     if not lbkr[lang] then goto next end
6943
6944     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6945     -- loops are nested.
6946     for k=1, #lbkr[lang] do
6947         local p = lbkr[lang][k].pattern
6948         local r = lbkr[lang][k].replace
6949         local attr = lbkr[lang][k].attr or -1
6950
6951         if Babel.debug then
6952             print('*****', p, mode)
6953         end
6954
6955         -- This variable is set in some cases below to the first *byte*
6956         -- after the match, either as found by u.match (faster) or the
6957         -- computed position based on sc if w has changed.
6958         local last_match = 0
6959         local step = 0
6960
6961         -- For every match.
6962         while true do
6963             if Babel.debug then
6964                 print('====')
6965             end
6966             local new -- used when inserting and removing nodes
6967
6968             local matches = { u.match(w, p, last_match) }
6969
6970             if #matches < 2 then break end
6971
6972             -- Get and remove empty captures (with ())'s, which return a
6973             -- number with the position), and keep actual captures
6974             -- (from (...)), if any, in matches.
6975             local first = table.remove(matches, 1)
6976             local last = table.remove(matches, #matches)
6977             -- Non re-fetched substrings may contain \31, which separates
6978             -- substrings.
6979             if string.find(w:sub(first, last-1), Babel.us_char) then break end
6980
6981             local save_last = last -- with A()BC()D, points to D
6982
6983             -- Fix offsets, from bytes to unicode. Explained above.
6984             first = u.len(w:sub(1, first-1)) + 1
6985             last = u.len(w:sub(1, last-1)) -- now last points to C
6986
6987             -- This loop stores in a small table the nodes
6988             -- corresponding to the pattern. Used by 'data' to provide a
6989             -- predictable behavior with 'insert' (w_nodes is modified on
6990             -- the fly), and also access to 'remove'd nodes.

```

```

6991     local sc = first-1          -- Used below, too
6992     local data_nodes = {}
6993
6994     local enabled = true
6995     for q = 1, last-first+1 do
6996         data_nodes[q] = w_nodes[sc+q]
6997         if enabled
6998             and attr > -1
6999             and not node.has_attribute(data_nodes[q], attr)
7000         then
7001             enabled = false
7002         end
7003     end
7004
7005     -- This loop traverses the matched substring and takes the
7006     -- corresponding action stored in the replacement list.
7007     -- sc = the position in substr nodes / string
7008     -- rc = the replacement table index
7009     local rc = 0
7010
7011     while rc < last-first+1 do -- for each replacement
7012         if Babel.debug then
7013             print('.....', rc + 1)
7014         end
7015         sc = sc + 1
7016         rc = rc + 1
7017
7018         if Babel.debug then
7019             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7020             local ss = ''
7021             for itt in node.traverse(head) do
7022                 if itt.id == 29 then
7023                     ss = ss .. unicode.utf8.char(itt.char)
7024                 else
7025                     ss = ss .. '{' .. itt.id .. '}'
7026                 end
7027             end
7028             print('*****', ss)
7029         end
7030
7031         local crep = r[rc]
7032         local item = w_nodes[sc]
7033         local item_base = item
7034         local placeholder = Babel.us_char
7035         local d
7036
7037         if crep and crep.data then
7038             item_base = data_nodes[crep.data]
7039         end
7040
7041         if crep then
7042             step = crep.step or 0
7043         end
7044
7045         if (not enabled) or (crep and next(crep) == nil) then -- = {}
7046             last_match = save_last    -- Optimization
7047             goto next
7048         end
7049
7050         elseif crep == nil or crep.remove then
7051             node.remove(head, item)
7052             table.remove(w_nodes, sc)
7053             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)

```

```

7054         sc = sc - 1 -- Nothing has been inserted.
7055         last_match = utf8.offset(w, sc+1+step)
7056         goto next
7057
7058     elseif crep and crep.kashida then -- Experimental
7059         node.set_attribute(item,
7060             Babel.attr_kashida,
7061             crep.kashida)
7062         last_match = utf8.offset(w, sc+1+step)
7063         goto next
7064
7065     elseif crep and crep.string then
7066         local str = crep.string(matches)
7067         if str == '' then -- Gather with nil
7068             node.remove(head, item)
7069             table.remove(w_nodes, sc)
7070             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
7071             sc = sc - 1 -- Nothing has been inserted.
7072         else
7073             local loop_first = true
7074             for s in string.utfvalues(str) do
7075                 d = node.copy(item_base)
7076                 d.char = s
7077                 if loop_first then
7078                     loop_first = false
7079                     head, new = node.insert_before(head, item, d)
7080                     if sc == 1 then
7081                         word_head = head
7082                     end
7083                     w_nodes[sc] = d
7084                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
7085                 else
7086                     sc = sc + 1
7087                     head, new = node.insert_before(head, item, d)
7088                     table.insert(w_nodes, sc, new)
7089                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
7090                 end
7091                 if Babel.debug then
7092                     print('.....', 'str')
7093                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7094                 end
7095             end -- for
7096             node.remove(head, item)
7097         end -- if ''
7098         last_match = utf8.offset(w, sc+1+step)
7099         goto next
7100
7101     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
7102         d = node.new(7, 3) -- (disc, regular)
7103         d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
7104         d.post = Babel.str_to_nodes(crep.post, matches, item_base)
7105         d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
7106         d.attr = item_base.attr
7107         if crep.pre == nil then -- TeXbook p96
7108             d.penalty = crep.penalty or tex.hyphenpenalty
7109         else
7110             d.penalty = crep.penalty or tex.exhyphenpenalty
7111         end
7112         placeholder = '|'
7113         head, new = node.insert_before(head, item, d)
7114
7115     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
7116         -- ERROR

```

```

7117
7118 elseif crep and crep.penalty then
7119     d = node.new(14, 0) -- (penalty, userpenalty)
7120     d.attr = item_base.attr
7121     d.penalty = crep.penalty
7122     head, new = node.insert_before(head, item, d)
7123
7124 elseif crep and crep.space then
7125     -- 655360 = 10 pt = 10 * 65536 sp
7126     d = node.new(12, 13) -- (glue, spaceskip)
7127     local quad = font.getfont(item_base.font).size or 655360
7128     node.setglue(d, crep.space[1] * quad,
7129                  crep.space[2] * quad,
7130                  crep.space[3] * quad)
7131     if mode == 0 then
7132         placeholder = ' '
7133     end
7134     head, new = node.insert_before(head, item, d)
7135
7136 elseif crep and crep.spacefactor then
7137     d = node.new(12, 13) -- (glue, spaceskip)
7138     local base_font = font.getfont(item_base.font)
7139     node.setglue(d,
7140                  crep.spacefactor[1] * base_font.parameters['space'],
7141                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
7142                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
7143     if mode == 0 then
7144         placeholder = ' '
7145     end
7146     head, new = node.insert_before(head, item, d)
7147
7148 elseif mode == 0 and crep and crep.space then
7149     -- ERROR
7150
7151 end -- ie replacement cases
7152
7153 -- Shared by disc, space and penalty.
7154 if sc == 1 then
7155     word_head = head
7156 end
7157 if crep.insert then
7158     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
7159     table.insert(w_nodes, sc, new)
7160     last = last + 1
7161 else
7162     w_nodes[sc] = d
7163     node.remove(head, item)
7164     w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
7165 end
7166
7167 last_match = utf8.offset(w, sc+1+step)
7168
7169 ::next::
7170
7171 end -- for each replacement
7172
7173 if Babel.debug then
7174     print('.....', '/')
7175     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
7176 end
7177
7178 end -- for match
7179

```



```

7180     end -- for patterns
7181
7182     ::next::
7183     word_head = nw
7184 end -- for substring
7185 return head
7186 end
7187
7188 -- This table stores capture maps, numbered consecutively
7189 Babel.capture_maps = {}
7190
7191 -- The following functions belong to the next macro
7192 function Babel.capture_func(key, cap)
7193     local ret = "[" .. cap:gsub('{{[0-9]}}', ")]..m[%1]..[" .. "]"
7194     local cnt
7195     local u = unicode.utf8
7196     ret, cnt = ret:gsub('{{[0-9]}|([^\]|+)|(.-)}', Babel.capture_func_map)
7197     if cnt == 0 then
7198         ret = u.gsub(ret, '{{(%x%x%x%x+)}',
7199             function (n)
7200                 return u.char(tonumber(n, 16))
7201             end)
7202     end
7203     ret = ret:gsub("%[%[%]]%.", '')
7204     ret = ret:gsub("%.%.%[%[%]]", '')
7205     return key .. [[=function(m) return ]] .. ret .. [[ end]]
7206 end
7207
7208 function Babel.capt_map(from, mapno)
7209     return Babel.capture_maps[mapno][from] or from
7210 end
7211
7212 -- Handle the {n|abc|ABC} syntax in captures
7213 function Babel.capture_func_map(capno, from, to)
7214     local u = unicode.utf8
7215     from = u.gsub(from, '{{(%x%x%x%x+)}',
7216         function (n)
7217             return u.char(tonumber(n, 16))
7218         end)
7219     to = u.gsub(to, '{{(%x%x%x%x+)}',
7220         function (n)
7221             return u.char(tonumber(n, 16))
7222         end)
7223     local froms = {}
7224     for s in string.utfcharacters(from) do
7225         table.insert(froms, s)
7226     end
7227     local cnt = 1
7228     table.insert(Babel.capture_maps, {})
7229     local mlen = table.getn(Babel.capture_maps)
7230     for s in string.utfcharacters(to) do
7231         Babel.capture_maps[mlen][froms[cnt]] = s
7232         cnt = cnt + 1
7233     end
7234     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
7235         (mlen) .. ").. " .. "["
7236 end
7237
7238 -- Create/Extend reversed sorted list of kashida weights:
7239 function Babel.capture_kashida(key, wt)
7240     wt = tonumber(wt)
7241     if Babel.kashida_wts then
7242         for p, q in ipairs(Babel.kashida_wts) do

```

```

7243     if wt == q then
7244         break
7245     elseif wt > q then
7246         table.insert(Babel.kashida_wts, p, wt)
7247         break
7248     elseif table.getn(Babel.kashida_wts) == p then
7249         table.insert(Babel.kashida_wts, wt)
7250     end
7251 end
7252 else
7253     Babel.kashida_wts = { wt }
7254 end
7255 return 'kashida = ' .. wt
7256 end
7257
7258 -- Experimental: applies prehyphenation transforms to a string (letters
7259 -- and spaces).
7260 function Babel.string_prehyphenation(str, locale)
7261     local n, head, last, res
7262     head = node.new(8, 0) -- dummy (hack just to start)
7263     last = head
7264     for s in string.utfvalues(str) do
7265         if s == 20 then
7266             n = node.new(12, 0)
7267         else
7268             n = node.new(29, 0)
7269             n.char = s
7270         end
7271         node.set_attribute(n, Babel.attr_locale, locale)
7272         last.next = n
7273         last = n
7274     end
7275     head = Babel.hyphenate_replace(head, 0)
7276     res = ''
7277     for n in node.traverse(head) do
7278         if n.id == 12 then
7279             res = res .. ' '
7280         elseif n.id == 29 then
7281             res = res .. unicode.utf8.char(n.char)
7282         end
7283     end
7284     tex.print(res)
7285 end
7286 </transforms>

```

10.12 Lua: Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is

still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidirectional.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 words, *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.

```

7287 (*basic-r)
7288 Babel = Babel or {}
7289
7290 Babel.bidi_enabled = true
7291
7292 require('babel-data-bidi.lua')
7293
7294 local characters = Babel.characters
7295 local ranges = Babel.ranges
7296
7297 local DIR = node.id("dir")
7298
7299 local function dir_mark(head, from, to, outer)
7300   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7301   local d = node.new(DIR)
7302   d.dir = '+' .. dir
7303   node.insert_before(head, from, d)
7304   d = node.new(DIR)
7305   d.dir = '-' .. dir
7306   node.insert_after(head, to, d)
7307 end
7308
7309 function Babel.bidi(head, ispar)
7310   local first_n, last_n          -- first and last char with nums
7311   local last_es                  -- an auxiliary 'last' used with nums
7312   local first_d, last_d          -- first and last char in L/R block
7313   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):

```

7314   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7315   local strong_lr = (strong == 'l') and 'l' or 'r'
7316   local outer = strong
7317
7318   local new_dir = false
7319   local first_dir = false
7320   local inmath = false
7321
7322   local last_lr
7323
7324   local type_n = ''
7325

```

```

7326 for item in node.traverse(head) do
7327
7328   -- three cases: glyph, dir, otherwise
7329   if item.id == node.id'glyph'
7330     or (item.id == 7 and item.subtype == 2) then
7331
7332     local itemchar
7333     if item.id == 7 and item.subtype == 2 then
7334       itemchar = item.replace.char
7335     else
7336       itemchar = item.char
7337     end
7338     local chardata = characters[itemchar]
7339     dir = chardata and chardata.d or nil
7340     if not dir then
7341       for nn, et in ipairs(ranges) do
7342         if itemchar < et[1] then
7343           break
7344         elseif itemchar <= et[2] then
7345           dir = et[3]
7346           break
7347         end
7348       end
7349     end
7350     dir = dir or 'l'
7351     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).

```

7352   if new_dir then
7353     attr_dir = 0
7354     for at in node.traverse(item.attr) do
7355       if at.number == Babel.attr_dir then
7356         attr_dir = at.value & 0x3
7357       end
7358     end
7359     if attr_dir == 1 then
7360       strong = 'r'
7361     elseif attr_dir == 2 then
7362       strong = 'al'
7363     else
7364       strong = 'l'
7365     end
7366     strong_lr = (strong == 'l') and 'l' or 'r'
7367     outer = strong_lr
7368     new_dir = false
7369   end
7370
7371   if dir == 'nsm' then dir = strong end -- W1

```

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

```

7372   dir_real = dir -- We need dir_real to set strong below
7373   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:

```

7374   if strong == 'al' then
7375     if dir == 'en' then dir = 'an' end -- W2
7376     if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7377     strong_lr = 'r' -- W3
7378   end

```

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

```

7379     elseif item.id == node.id'dir' and not inmath then
7380         new_dir = true
7381         dir = nil
7382     elseif item.id == node.id'math' then
7383         inmath = (item.subtype == 0)
7384     else
7385         dir = nil          -- Not a char
7386     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>.

```

7387     if dir == 'en' or dir == 'an' or dir == 'et' then
7388         if dir ~= 'et' then
7389             type_n = dir
7390         end
7391         first_n = first_n or item
7392         last_n = last_es or item
7393         last_es = nil
7394     elseif dir == 'es' and last_n then -- W3+W6
7395         last_es = item
7396     elseif dir == 'cs' then          -- it's right - do nothing
7397     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7398         if strong_lr == 'r' and type_n ~= '' then
7399             dir_mark(head, first_n, last_n, 'r')
7400         elseif strong_lr == 'l' and first_d and type_n == 'an' then
7401             dir_mark(head, first_n, last_n, 'r')
7402             dir_mark(head, first_d, last_d, outer)
7403             first_d, last_d = nil, nil
7404         elseif strong_lr == 'l' and type_n ~= '' then
7405             last_d = last_n
7406         end
7407         type_n = ''
7408         first_n, last_n = nil, nil
7409     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:

```

7410     if dir == 'l' or dir == 'r' then
7411         if dir ~= outer then
7412             first_d = first_d or item
7413             last_d = item
7414         elseif first_d and dir ~= strong_lr then
7415             dir_mark(head, first_d, last_d, outer)
7416             first_d, last_d = nil, nil
7417         end
7418     end

```

Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resp'tly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

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

```

7419     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7420         item.char = characters[item.char] and
7421             characters[item.char].m or item.char
7422     elseif (dir or new_dir) and last_lr ~= item then
7423         local mir = outer .. strong_lr .. (dir or outer)

```

```

7424     if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7425         for ch in node.traverse(node.next(last_lr)) do
7426             if ch == item then break end
7427             if ch.id == node.id'glyph' and characters[ch.char] then
7428                 ch.char = characters[ch.char].m or ch.char
7429             end
7430         end
7431     end
7432 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).

```

7433     if dir == 'l' or dir == 'r' then
7434         last_lr = item
7435         strong = dir_real          -- Don't search back - best save now
7436         strong_lr = (strong == 'l') and 'l' or 'r'
7437     elseif new_dir then
7438         last_lr = nil
7439     end
7440 end

```

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

```

7441     if last_lr and outer == 'r' then
7442         for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7443             if characters[ch.char] then
7444                 ch.char = characters[ch.char].m or ch.char
7445             end
7446         end
7447     end
7448     if first_n then
7449         dir_mark(head, first_n, last_n, outer)
7450     end
7451     if first_d then
7452         dir_mark(head, first_d, last_d, outer)
7453     end

```

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

```

7454     return node.prev(head) or head
7455 end
7456 </basic-r>

```

And here the Lua code for bidi=basic:

```

7457 <(*basic)
7458 Babel = Babel or {}
7459
7460 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7461
7462 Babel.fontmap = Babel.fontmap or {}
7463 Babel.fontmap[0] = {}          -- l
7464 Babel.fontmap[1] = {}          -- r
7465 Babel.fontmap[2] = {}          -- al/an
7466
7467 Babel.bidi_enabled = true
7468 Babel.mirroring_enabled = true
7469
7470 require('babel-data-bidi.lua')
7471
7472 local characters = Babel.characters
7473 local ranges = Babel.ranges
7474
7475 local DIR = node.id('dir')
7476 local GLYPH = node.id('glyph')
7477

```

```

7478 local function insert_implicit(head, state, outer)
7479   local new_state = state
7480   if state.sim and state.eim and state.sim ~= state.eim then
7481     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7482     local d = node.new(DIR)
7483     d.dir = '+' .. dir
7484     node.insert_before(head, state.sim, d)
7485     local d = node.new(DIR)
7486     d.dir = '-' .. dir
7487     node.insert_after(head, state.eim, d)
7488   end
7489   new_state.sim, new_state.eim = nil, nil
7490   return head, new_state
7491 end
7492
7493 local function insert_numeric(head, state)
7494   local new
7495   local new_state = state
7496   if state.san and state.ean and state.san ~= state.ean then
7497     local d = node.new(DIR)
7498     d.dir = '+TLT'
7499     _, new = node.insert_before(head, state.san, d)
7500     if state.san == state.sim then state.sim = new end
7501     local d = node.new(DIR)
7502     d.dir = '-TLT'
7503     _, new = node.insert_after(head, state.ean, d)
7504     if state.ean == state.eim then state.eim = new end
7505   end
7506   new_state.san, new_state.ean = nil, nil
7507   return head, new_state
7508 end
7509
7510 -- TODO - \hbox with an explicit dir can lead to wrong results
7511 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7512 -- was s made to improve the situation, but the problem is the 3-dir
7513 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7514 -- well.
7515
7516 function Babel.bidi(head, ispar, hdir)
7517   local d -- d is used mainly for computations in a loop
7518   local prev_d = ''
7519   local new_d = false
7520
7521   local nodes = {}
7522   local outer_first = nil
7523   local inmath = false
7524
7525   local glue_d = nil
7526   local glue_i = nil
7527
7528   local has_en = false
7529   local first_et = nil
7530
7531   local has_hyperlink = false
7532
7533   local ATDIR = Babel.attr_dir
7534
7535   local save_outer
7536   local temp = node.get_attribute(head, ATDIR)
7537   if temp then
7538     temp = temp & 0x3
7539     save_outer = (temp == 0 and 'l') or
7540                  (temp == 1 and 'r') or

```

```

7541             (temp == 2 and 'al')
7542 elseif ispar then             -- Or error? Shouldn't happen
7543     save_outter = ('TRT' == tex.pardir) and 'r' or 'l'
7544 else                         -- Or error? Shouldn't happen
7545     save_outter = ('TRT' == hdir) and 'r' or 'l'
7546 end
7547 -- when the callback is called, we are just _after_ the box,
7548 -- and the textdir is that of the surrounding text
7549 -- if not ispar and hdir ~= tex.textdir then
7550 --     save_outter = ('TRT' == hdir) and 'r' or 'l'
7551 -- end
7552 local outer = save_outter
7553 local last = outer
7554 -- 'al' is only taken into account in the first, current loop
7555 if save_outter == 'al' then save_outter = 'r' end
7556
7557 local fontmap = Babel.fontmap
7558
7559 for item in node.traverse(head) do
7560
7561     -- In what follows, #node is the last (previous) node, because the
7562     -- current one is not added until we start processing the neutrals.
7563
7564     -- three cases: glyph, dir, otherwise
7565     if item.id == GLYPH
7566         or (item.id == 7 and item.subtype == 2) then
7567
7568         local d_font = nil
7569         local item_r
7570         if item.id == 7 and item.subtype == 2 then
7571             item_r = item.replace -- automatic discs have just 1 glyph
7572         else
7573             item_r = item
7574         end
7575         local chardata = characters[item_r.char]
7576         d = chardata and chardata.d or nil
7577         if not d or d == 'nsm' then
7578             for nn, et in ipairs(ranges) do
7579                 if item_r.char < et[1] then
7580                     break
7581                 elseif item_r.char <= et[2] then
7582                     if not d then d = et[3]
7583                     elseif d == 'nsm' then d_font = et[3]
7584                     end
7585                     break
7586                 end
7587             end
7588         end
7589         d = d or 'l'
7590
7591         -- A short 'pause' in bidi for mapfont
7592         d_font = d_font or d
7593         d_font = (d_font == 'l' and 0) or
7594             (d_font == 'nsm' and 0) or
7595             (d_font == 'r' and 1) or
7596             (d_font == 'al' and 2) or
7597             (d_font == 'an' and 2) or nil
7598         if d_font and fontmap and fontmap[d_font][item_r.font] then
7599             item_r.font = fontmap[d_font][item_r.font]
7600         end
7601
7602         if new_d then
7603             table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})

```



```

7604     if inmath then
7605         attr_d = 0
7606     else
7607         attr_d = node.get_attribute(item, ATDIR)
7608         attr_d = attr_d & 0x3
7609     end
7610     if attr_d == 1 then
7611         outer_first = 'r'
7612         last = 'r'
7613     elseif attr_d == 2 then
7614         outer_first = 'r'
7615         last = 'al'
7616     else
7617         outer_first = 'l'
7618         last = 'l'
7619     end
7620     outer = last
7621     has_en = false
7622     first_et = nil
7623     new_d = false
7624 end
7625
7626 if glue_d then
7627     if (d == 'l' and 'l' or 'r') ~= glue_d then
7628         table.insert(nodes, {glue_i, 'on', nil})
7629     end
7630     glue_d = nil
7631     glue_i = nil
7632 end
7633
7634 elseif item.id == DIR then
7635     d = nil
7636
7637     if head ~= item then new_d = true end
7638
7639 elseif item.id == node.id'glue' and item.subtype == 13 then
7640     glue_d = d
7641     glue_i = item
7642     d = nil
7643
7644 elseif item.id == node.id'math' then
7645     inmath = (item.subtype == 0)
7646
7647 elseif item.id == 8 and item.subtype == 19 then
7648     has_hyperlink = true
7649
7650 else
7651     d = nil
7652 end
7653
7654 -- AL <= EN/ET/ES      -- W2 + W3 + W6
7655 if last == 'al' and d == 'en' then
7656     d = 'an'           -- W3
7657 elseif last == 'al' and (d == 'et' or d == 'es') then
7658     d = 'on'           -- W6
7659 end
7660
7661 -- EN + CS/ES + EN      -- W4
7662 if d == 'en' and #nodes >= 2 then
7663     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7664         and nodes[#nodes-1][2] == 'en' then
7665         nodes[#nodes][2] = 'en'
7666     end

```

```

7667     end
7668
7669     -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7670     if d == 'an' and #nodes >= 2 then
7671         if (nodes[#nodes][2] == 'cs')
7672             and nodes[#nodes-1][2] == 'an' then
7673             nodes[#nodes][2] = 'an'
7674         end
7675     end
7676
7677     -- ET/EN                  -- W5 + W7->l / W6->on
7678     if d == 'et' then
7679         first_et = first_et or (#nodes + 1)
7680     elseif d == 'en' then
7681         has_en = true
7682         first_et = first_et or (#nodes + 1)
7683     elseif first_et then      -- d may be nil here !
7684         if has_en then
7685             if last == 'l' then
7686                 temp = 'l'    -- W7
7687             else
7688                 temp = 'en'   -- W5
7689             end
7690         else
7691             temp = 'on'       -- W6
7692         end
7693         for e = first_et, #nodes do
7694             if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7695         end
7696         first_et = nil
7697         has_en = false
7698     end
7699
7700     -- Force mathdir in math if ON (currently works as expected only
7701     -- with 'l')
7702     if inmath and d == 'on' then
7703         d = ('TRT' == tex.mathdir) and 'r' or 'l'
7704     end
7705
7706     if d then
7707         if d == 'al' then
7708             d = 'r'
7709             last = 'al'
7710         elseif d == 'l' or d == 'r' then
7711             last = d
7712         end
7713         prev_d = d
7714         table.insert(nodes, {item, d, outer_first})
7715     end
7716
7717     outer_first = nil
7718
7719 end
7720
7721 -- TODO -- repeated here in case EN/ET is the last node. Find a
7722 -- better way of doing things:
7723 if first_et then      -- dir may be nil here !
7724     if has_en then
7725         if last == 'l' then
7726             temp = 'l'    -- W7
7727         else
7728             temp = 'en'   -- W5
7729         end

```

```

7730     else
7731         temp = 'on'      -- W6
7732     end
7733     for e = first_et, #nodes do
7734         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7735     end
7736 end
7737
7738 -- dummy node, to close things
7739 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7740
7741 ----- NEUTRAL -----
7742
7743 outer = save_outer
7744 last = outer
7745
7746 local first_on = nil
7747
7748 for q = 1, #nodes do
7749     local item
7750
7751     local outer_first = nodes[q][3]
7752     outer = outer_first or outer
7753     last = outer_first or last
7754
7755     local d = nodes[q][2]
7756     if d == 'an' or d == 'en' then d = 'r' end
7757     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7758
7759     if d == 'on' then
7760         first_on = first_on or q
7761     elseif first_on then
7762         if last == d then
7763             temp = d
7764         else
7765             temp = outer
7766         end
7767         for r = first_on, q - 1 do
7768             nodes[r][2] = temp
7769             item = nodes[r][1]      -- MIRRORING
7770             if Babel.mirroring_enabled and item.id == GLYPH
7771                 and temp == 'r' and characters[item.char] then
7772                 local font_mode = ''
7773                 if item.font > 0 and font.fonts[item.font].properties then
7774                     font_mode = font.fonts[item.font].properties.mode
7775                 end
7776                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
7777                     item.char = characters[item.char].m or item.char
7778                 end
7779             end
7780         end
7781         first_on = nil
7782     end
7783
7784     if d == 'r' or d == 'l' then last = d end
7785 end
7786
7787 ----- IMPLICIT, REORDER -----
7788
7789 outer = save_outer
7790 last = outer
7791
7792 local state = {}

```

```

7793 state.has_r = false
7794
7795 for q = 1, #nodes do
7796     local item = nodes[q][1]
7797
7798     outer = nodes[q][3] or outer
7799
7800     local d = nodes[q][2]
7801
7802     if d == 'nsm' then d = last end          -- W1
7803     if d == 'en' then d = 'an' end
7804     local isdir = (d == 'r' or d == 'l')
7805
7806     if outer == 'l' and d == 'an' then
7807         state.san = state.san or item
7808         state.ean = item
7809     elseif state.san then
7810         head, state = insert_numeric(head, state)
7811     end
7812
7813     if outer == 'l' then
7814         if d == 'an' or d == 'r' then      -- im -> implicit
7815             if d == 'r' then state.has_r = true end
7816             state.sim = state.sim or item
7817             state.eim = item
7818         elseif d == 'l' and state.sim and state.has_r then
7819             head, state = insert_implicit(head, state, outer)
7820         elseif d == 'l' then
7821             state.sim, state.eim, state.has_r = nil, nil, false
7822         end
7823     else
7824         if d == 'an' or d == 'l' then
7825             if nodes[q][3] then -- nil except after an explicit dir
7826                 state.sim = item -- so we move sim 'inside' the group
7827             else
7828                 state.sim = state.sim or item
7829             end
7830             state.eim = item
7831         elseif d == 'r' and state.sim then
7832             head, state = insert_implicit(head, state, outer)
7833         elseif d == 'r' then
7834             state.sim, state.eim = nil, nil
7835         end
7836     end
7837 end
7838
7839 if isdir then
7840     last = d          -- Don't search back - best save now
7841 elseif d == 'on' and state.san then
7842     state.san = state.san or item
7843     state.ean = item
7844 end
7845
7846 end
7847
7848 head = node.prev(head) or head
7849
7850 ----- FIX HYPERLINKS -----
7851
7852 if has_hyperlink then
7853     local flag, linking = 0, 0
7854     for item in node.traverse(head) do
7855         if item.id == DIR then

```

```

7856         if item.dir == '+TRT' or item.dir == '+TLT' then
7857             flag = flag + 1
7858         elseif item.dir == '-TRT' or item.dir == '-TLT' then
7859             flag = flag - 1
7860         end
7861     elseif item.id == 8 and item.subtype == 19 then
7862         linking = flag
7863     elseif item.id == 8 and item.subtype == 20 then
7864         if linking > 0 then
7865             if item.prev.id == DIR and
7866                 (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7867                 d = node.new(DIR)
7868                 d.dir = item.prev.dir
7869                 node.remove(head, item.prev)
7870                 node.insert_after(head, item, d)
7871             end
7872         end
7873         linking = 0
7874     end
7875 end
7876 end
7877
7878 return head
7879 end
7880 </basic>

```

11 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},

```

For the meaning of these codes, see the Unicode standard.

12 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```

7881 <*nil>
7882 \ProvidesLanguage{nil}[<<date>> v<<version>> Nil language]
7883 \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.

```

7884 \ifx\l@nil\undefined
7885   \newlanguage\l@nil
7886   \@namedef{bbl@hyphendata@the\l@nil}{{}}% Remove warning
7887   \let\bbl@elt\relax
7888   \edef\bbl@languages{% Add it to the list of languages
7889     \bbl@languages\bbl@elt{nil}{the\l@nil}{{}}
7890 \fi

```

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

```

7891 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

```

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

```
\captionnil
\datenil
7892 \let\captionsnil\@empty
7893 \let\datenil\@empty
```

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

```
7894 \def\bbl@inidata@nil{%
7895   \bbl@elt{identification}{tag.ini}{und}%
7896   \bbl@elt{identification}{load.level}{0}%
7897   \bbl@elt{identification}{charset}{utf8}%
7898   \bbl@elt{identification}{version}{1.0}%
7899   \bbl@elt{identification}{date}{2022-05-16}%
7900   \bbl@elt{identification}{name.local}{nil}%
7901   \bbl@elt{identification}{name.english}{nil}%
7902   \bbl@elt{identification}{name.babel}{nil}%
7903   \bbl@elt{identification}{tag.bcp47}{und}%
7904   \bbl@elt{identification}{language.tag.bcp47}{und}%
7905   \bbl@elt{identification}{tag.opentype}{dflt}%
7906   \bbl@elt{identification}{script.name}{Latin}%
7907   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7908   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7909   \bbl@elt{identification}{level}{1}%
7910   \bbl@elt{identification}{encodings}{}%
7911   \bbl@elt{identification}{derivate}{no}}
7912 \@namedef{bbl@tbc@nil}{und}
7913 \@namedef{bbl@lbc@nil}{und}
7914 \@namedef{bbl@casing@nil}{und} % TODO
7915 \@namedef{bbl@lotf@nil}{dflt}
7916 \@namedef{bbl@elname@nil}{nil}
7917 \@namedef{bbl@lname@nil}{nil}
7918 \@namedef{bbl@esname@nil}{Latin}
7919 \@namedef{bbl@sname@nil}{Latin}
7920 \@namedef{bbl@sbc@nil}{Latn}
7921 \@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.

```
7922 \ldf@finish{nil}
7923 \</nil>
```

13 Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the identification section with `require.calendars`.

Start with function to compute the Julian day. It’s based on the little library `calendar.js`, by John Walker, in the public domain.

```
7924 <<Compute Julian day>> ≡
7925 \def\bbl@fpmmod#1#2{(#1-#2*floor(#1/#2))}
7926 \def\bbl@cs@gregleap#1{%
7927   (\bbl@fpmmod{#1}{4} == 0) &&
7928   (!((\bbl@fpmmod{#1}{100} == 0) && (\bbl@fpmmod{#1}{400} != 0)))}
7929 \def\bbl@cs@jd#1#2#3{% year, month, day
7930   \fp_eval:n{ 1721424.5 + (365 * (#1 - 1)) +
7931     floor((#1 - 1) / 4) + (-floor((#1 - 1) / 100)) +
7932     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7933     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7934 <</Compute Julian day>>
```

13.1 Islamic

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

```

7935 (*ca-islamic)
7936 \ExplSyntaxOn
7937 <<Compute Julian day>>
7938 % == islamic (default)
7939 % Not yet implemented
7940 \def\bbl@ca@islamic#1-#2-#3\@#4#5#6{

```

The Civil calendar.

```

7941 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7942 ((#3 + ceil(29.5 * (#2 - 1)) +
7943 (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7944 1948439.5) - 1) }
7945 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7946 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7947 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7948 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7949 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7950 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@#5#6#7{%
7951 \edef\bbl@tempa{%
7952 \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7953 \edef#5{%
7954 \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7955 \edef#6{\fp_eval:n{
7956 min(12, ceil((\bbl@tempa - (29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7957 \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).

```

7958 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7959 56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7960 57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7961 57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7962 57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7963 58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7964 58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7965 58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7966 58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7967 59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7968 59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7969 59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7970 60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7971 60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7972 60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7973 60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7974 61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7975 61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7976 61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7977 62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7978 62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7979 62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7980 63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7981 63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7982 63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7983 63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7984 64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7985 64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7986 64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7987 65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7988 65401,65431,65460,65490,65520}
7989 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7990 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7991 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}

```

```

7992 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@#5#6#7{%
7993   \ifnum#2>2014 \ifnum#2<2038
7994     \bbl@afterfi\expandafter\@gobble
7995   \fi\fi
7996   {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}}%
7997 \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7998   \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7999 \count@ \@ne
8000 \bbl@foreach\bbl@cs@umalqura@data{%
8001   \advance\count@ \@ne
8002   \ifnum##1>\bbl@tempd\else
8003     \edef\bbl@tempe{\the\count@}%
8004     \edef\bbl@tempb{##1}%
8005   \fi}%
8006 \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
8007 \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
8008 \edef#5{\fp_eval:n{ \bbl@tempa + 1 }}%
8009 \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
8010 \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}%
8011 \ExplSyntaxOff
8012 \bbl@add\bbl@precalendar{%
8013   \bbl@replace\bbl@ld@calendar{-civil}}}%
8014   \bbl@replace\bbl@ld@calendar{-umalqura}}}%
8015   \bbl@replace\bbl@ld@calendar{+}}}%
8016   \bbl@replace\bbl@ld@calendar{-}}}%
8017 \</ca-islamic>

```

13.2 Hebrew

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

```

8018 \*ca-hebrew)
8019 \newcount\bbl@cntcommon
8020 \def\bbl@remainder#1#2#3{%
8021   #3=#1\relax
8022   \divide #3 by #2\relax
8023   \multiply #3 by -#2\relax
8024   \advance #3 by #1\relax}%
8025 \newif\ifbbl@divisible
8026 \def\bbl@checkifdivisible#1#2{%
8027   {\countdef\tmp=0
8028     \bbl@remainder{#1}{#2}{\tmp}%
8029     \ifnum \tmp=0
8030       \global\bbl@divisibletrue
8031     \else
8032       \global\bbl@divisiblefalse
8033     \fi}}
8034 \newif\ifbbl@gregleap
8035 \def\bbl@ifgregleap#1{%
8036   \bbl@checkifdivisible{#1}{4}%
8037   \ifbbl@divisible
8038     \bbl@checkifdivisible{#1}{100}%
8039     \ifbbl@divisible
8040       \bbl@checkifdivisible{#1}{400}%
8041       \ifbbl@divisible
8042         \bbl@gregleaptrue
8043       \else
8044         \bbl@gregleapfalse
8045       \fi
8046     \else
8047       \bbl@gregleaptrue
8048     \fi

```



```

8049 \else
8050     \bbl@gregleapfalse
8051 \fi
8052 \ifbbl@gregleap}
8053 \def\bbl@gregdayspriormonths#1#2#3{%
8054     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
8055         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
8056     \bbl@ifgregleap{#2}%
8057     \ifnum #1 > 2
8058         \advance #3 by 1
8059     \fi
8060 \fi
8061 \global\bbl@cntcommon=#3}%
8062 #3=\bbl@cntcommon}
8063 \def\bbl@gregdaysprioryears#1#2{%
8064     {\countdef\tmpc=4
8065     \countdef\tmpb=2
8066     \tmpb=#1\relax
8067     \advance \tmpb by -1
8068     \tmpc=\tmpb
8069     \multiply \tmpc by 365
8070     #2=\tmpc
8071     \tmpc=\tmpb
8072     \divide \tmpc by 4
8073     \advance #2 by \tmpc
8074     \tmpc=\tmpb
8075     \divide \tmpc by 100
8076     \advance #2 by -\tmpc
8077     \tmpc=\tmpb
8078     \divide \tmpc by 400
8079     \advance #2 by \tmpc
8080     \global\bbl@cntcommon=#2\relax}%
8081 #2=\bbl@cntcommon}
8082 \def\bbl@absfromgreg#1#2#3#4{%
8083     {\countdef\tmpd=0
8084     #4=#1\relax
8085     \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
8086     \advance #4 by \tmpd
8087     \bbl@gregdaysprioryears{#3}{\tmpd}%
8088     \advance #4 by \tmpd
8089     \global\bbl@cntcommon=#4\relax}%
8090 #4=\bbl@cntcommon}
8091 \newif\ifbbl@hebrleap
8092 \def\bbl@checkleaphebrewyear#1{%
8093     {\countdef\tmpa=0
8094     \countdef\tmpb=1
8095     \tmpa=#1\relax
8096     \multiply \tmpa by 7
8097     \advance \tmpa by 1
8098     \bbl@remainder{\tmpa}{19}{\tmpb}%
8099     \ifnum \tmpb < 7
8100         \global\bbl@hebrleaptrue
8101     \else
8102         \global\bbl@hebrleapfalse
8103     \fi}}
8104 \def\bbl@hebreleapsedmonths#1#2{%
8105     {\countdef\tmpa=0
8106     \countdef\tmpb=1
8107     \countdef\tmpc=2
8108     \tmpa=#1\relax
8109     \advance \tmpa by -1
8110     #2=\tmpa
8111     \divide #2 by 19

```

```

8112 \multiply #2 by 235
8113 \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
8114 \tmpc=\tmpb
8115 \multiply \tmpb by 12
8116 \advance #2 by \tmpb
8117 \multiply \tmpc by 7
8118 \advance \tmpc by 1
8119 \divide \tmpc by 19
8120 \advance #2 by \tmpc
8121 \global\bbl@cntcommon=#2}%
8122 #2=\bbl@cntcommon}
8123 \def\bbl@hebreleapseddays#1#2{%
8124 {\countdef\tmpa=0
8125 \countdef\tmpb=1
8126 \countdef\tmpc=2
8127 \bbl@hebreleapsedmonths{#1}{#2}%
8128 \tmpa=#2\relax
8129 \multiply \tmpa by 13753
8130 \advance \tmpa by 5604
8131 \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
8132 \divide \tmpa by 25920
8133 \multiply #2 by 29
8134 \advance #2 by 1
8135 \advance #2 by \tmpa
8136 \bbl@remainder{#2}{7}{\tmpa}%
8137 \ifnum \tmpc < 19440
8138 \ifnum \tmpc < 9924
8139 \else
8140 \ifnum \tmpa=2
8141 \bbl@checkleaphebrewyear{#1}% of a common year
8142 \ifbbl@hebrleap
8143 \else
8144 \advance #2 by 1
8145 \fi
8146 \fi
8147 \fi
8148 \ifnum \tmpc < 16789
8149 \else
8150 \ifnum \tmpa=1
8151 \advance #1 by -1
8152 \bbl@checkleaphebrewyear{#1}% at the end of leap year
8153 \ifbbl@hebrleap
8154 \advance #2 by 1
8155 \fi
8156 \fi
8157 \fi
8158 \else
8159 \advance #2 by 1
8160 \fi
8161 \bbl@remainder{#2}{7}{\tmpa}%
8162 \ifnum \tmpa=0
8163 \advance #2 by 1
8164 \else
8165 \ifnum \tmpa=3
8166 \advance #2 by 1
8167 \else
8168 \ifnum \tmpa=5
8169 \advance #2 by 1
8170 \fi
8171 \fi
8172 \fi
8173 \global\bbl@cntcommon=#2\relax}%
8174 #2=\bbl@cntcommon}

```

```

8175 \def\bbl@daysinhebrewyear#1#2{%
8176   {\countdef\tmpe=12
8177    \bbl@hebreleapseddays{#1}{\tmpe}%
8178    \advance #1 by 1
8179    \bbl@hebreleapseddays{#1}{#2}%
8180    \advance #2 by -\tmpe
8181    \global\bbl@cntcommon=#2}%
8182   #2=\bbl@cntcommon}
8183 \def\bbl@hebrdayspriormonths#1#2#3{%
8184   {\countdef\tmpf= 14
8185    #3=\ifcase #1\relax
8186      0 \or
8187      0 \or
8188      30 \or
8189      59 \or
8190      89 \or
8191      118 \or
8192      148 \or
8193      148 \or
8194      177 \or
8195      207 \or
8196      236 \or
8197      266 \or
8198      295 \or
8199      325 \or
8200      400
8201    \fi
8202    \bbl@checkleaphebrewyear{#2}%
8203    \ifbbl@hebrleap
8204      \ifnum #1 > 6
8205        \advance #3 by 30
8206      \fi
8207    \fi
8208    \bbl@daysinhebrewyear{#2}{\tmpf}%
8209    \ifnum #1 > 3
8210      \ifnum \tmpf=353
8211        \advance #3 by -1
8212      \fi
8213      \ifnum \tmpf=383
8214        \advance #3 by -1
8215      \fi
8216    \fi
8217    \ifnum #1 > 2
8218      \ifnum \tmpf=355
8219        \advance #3 by 1
8220      \fi
8221      \ifnum \tmpf=385
8222        \advance #3 by 1
8223      \fi
8224    \fi
8225    \global\bbl@cntcommon=#3\relax}%
8226   #3=\bbl@cntcommon}
8227 \def\bbl@absfromhebr#1#2#3#4{%
8228   {#4=#1\relax
8229    \bbl@hebrdayspriormonths{#2}{#3}{#1}%
8230    \advance #4 by #1\relax
8231    \bbl@hebreleapseddays{#3}{#1}%
8232    \advance #4 by #1\relax
8233    \advance #4 by -1373429
8234    \global\bbl@cntcommon=#4\relax}%
8235   #4=\bbl@cntcommon}
8236 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
8237   {\countdef\tmpx= 17

```

```

8238 \countdef\tmpy= 18
8239 \countdef\tmpz= 19
8240 #6=#3\relax
8241 \global\advance #6 by 3761
8242 \bbl@absfromgreg{#1}{#2}{#3}{#4}%
8243 \tmpz=1 \tmpy=1
8244 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8245 \ifnum \tmpx > #4\relax
8246 \global\advance #6 by -1
8247 \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
8248 \fi
8249 \advance #4 by -\tmpx
8250 \advance #4 by 1
8251 #5=#4\relax
8252 \divide #5 by 30
8253 \loop
8254 \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8255 \ifnum \tmpx < #4\relax
8256 \advance #5 by 1
8257 \tmpy=\tmpx
8258 \repeat
8259 \global\advance #5 by -1
8260 \global\advance #4 by -\tmpy}}
8261 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebyear
8262 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8263 \def\bbl@ca@hebrew#1-#2-#3\@#4#5#6{%
8264 \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8265 \bbl@hebrfromgreg
8266 {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8267 {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebyear}%
8268 \edef#4{\the\bbl@hebyear}%
8269 \edef#5{\the\bbl@hebrmonth}%
8270 \edef#6{\the\bbl@hebrday}}
8271 \ca-hebrew

```

13.3 Persian

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

```

8272 (*ca-persian)
8273 \ExplSyntaxOn
8274 <<Compute Julian day>>
8275 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8276 2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8277 \def\bbl@ca@persian#1-#2-#3\@#4#5#6{%
8278 \edef\bbl@tempa{#1}% 20XX-03-\bbl@tempe = 1 farvardin:
8279 \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8280 \bbl@afterfi\expandafter@gobble
8281 \fi\fi
8282 {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}}%
8283 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8284 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8285 \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8286 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8287 \ifnum\bbl@tempc<\bbl@tempb
8288 \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8289 \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8290 \ifin@def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8291 \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8292 \fi

```

```

8293 \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8294 \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8295 \edef#5{\fp_eval:n{% set Jalali month
8296   (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8297 \edef#6{\fp_eval:n{% set Jalali day
8298   (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6))}}}}
8299 \ExplSyntaxOff
8300 \ca-persian)

```

13.4 Coptic and Ethiopic

Adapted from `jquery.calendars.package-1.1.4`, written by Keith Wood, 2010. Dual license: GPL and MIT. The only difference is the epoch.

```

8301 (*ca-coptic)
8302 \ExplSyntaxOn
8303 <<Compute Julian day>>
8304 \def\bbl@ca@coptic#1-#2-#3\@#4#5#6{%
8305   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8306   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8307   \edef#4{\fp_eval:n{%
8308     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8309   \edef\bbl@tempc{\fp_eval:n{%
8310     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8311   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8312   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8313 \ExplSyntaxOff
8314 \ca-coptic)
8315 (*ca-ethiopic)
8316 \ExplSyntaxOn
8317 <<Compute Julian day>>
8318 \def\bbl@ca@ethiopic#1-#2-#3\@#4#5#6{%
8319   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8320   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8321   \edef#4{\fp_eval:n{%
8322     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8323   \edef\bbl@tempc{\fp_eval:n{%
8324     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8325   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8326   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}}
8327 \ExplSyntaxOff
8328 \ca-ethiopic)

```

13.5 Buddhist

That's very simple.

```

8329 (*ca-buddhist)
8330 \def\bbl@ca@buddhist#1-#2-#3\@#4#5#6{%
8331   \edef#4{\number\numexpr#1+543\relax}%
8332   \edef#5{#2}%
8333   \edef#6{#3}}
8334 \ca-buddhist)
8335 %
8336 % \subsection{Chinese}
8337 %
8338 % Brute force, with the Julian day of first day of each month. The
8339 % table has been computed with the help of \textsf{python-lunardate} by
8340 % Ricky Yeung, GPLv2 (but the code itself has not been used). The range
8341 % is 2015-2044.
8342 %
8343 % \begin{macrocode}
8344 (*ca-chinese)
8345 \ExplSyntaxOn

```

```

8346 <<Compute Julian day>>
8347 \def\bbl@ca@chinese#1-#2-#3\@#4#5#6{%
8348   \edef\bbl@tempd{\fp_eval:n{%
8349     \bbl@cs@jd{#1}{#2}{#3} - 2457072.5 }}%
8350   \count@z@
8351   \@tempcnta=2015
8352   \bbl@foreach\bbl@cs@chinese@data{%
8353     \ifnum##1>\bbl@tempd\else
8354       \advance\count@z@ne
8355       \ifnum\count@z@>12
8356         \count@z@ne
8357         \advance\@tempcnta\@ne\fi
8358       \bbl@xin@{,##1,}{,\bbl@cs@chinese@leap,}%
8359       \ifin@
8360         \advance\count@m@ne
8361         \edef\bbl@tempe{\the\numexpr\count@z@+12\relax}%
8362       \else
8363         \edef\bbl@tempe{\the\count@z@}%
8364       \fi
8365       \edef\bbl@tempb{##1}%
8366     \fi}%
8367   \edef#4{\the\@tempcnta}%
8368   \edef#5{\bbl@tempe}%
8369   \edef#6{\the\numexpr\bbl@tempd-\bbl@tempb+1\relax}}
8370 \def\bbl@cs@chinese@leap{%
8371   885,1920,2953,3809,4873,5906,6881,7825,8889,9893,10778}
8372 \def\bbl@cs@chinese@data{0,29,59,88,117,147,176,206,236,266,295,325,
8373   354,384,413,443,472,501,531,560,590,620,649,679,709,738,%
8374   768,797,827,856,885,915,944,974,1003,1033,1063,1093,1122,%
8375   1152,1181,1211,1240,1269,1299,1328,1358,1387,1417,1447,1477,%
8376   1506,1536,1565,1595,1624,1653,1683,1712,1741,1771,1801,1830,%
8377   1860,1890,1920,1949,1979,2008,2037,2067,2096,2126,2155,2185,%
8378   2214,2244,2274,2303,2333,2362,2392,2421,2451,2480,2510,2539,%
8379   2569,2598,2628,2657,2687,2717,2746,2776,2805,2835,2864,2894,%
8380   2923,2953,2982,3011,3041,3071,3100,3130,3160,3189,3219,3248,%
8381   3278,3307,3337,3366,3395,3425,3454,3484,3514,3543,3573,3603,%
8382   3632,3662,3691,3721,3750,3779,3809,3838,3868,3897,3927,3957,%
8383   3987,4016,4046,4075,4105,4134,4163,4193,4222,4251,4281,4311,%
8384   4341,4370,4400,4430,4459,4489,4518,4547,4577,4606,4635,4665,%
8385   4695,4724,4754,4784,4814,4843,4873,4902,4931,4961,4990,5019,%
8386   5049,5079,5108,5138,5168,5197,5227,5256,5286,5315,5345,5374,%
8387   5403,5433,5463,5492,5522,5551,5581,5611,5640,5670,5699,5729,%
8388   5758,5788,5817,5846,5876,5906,5935,5965,5994,6024,6054,6083,%
8389   6113,6142,6172,6201,6231,6260,6289,6319,6348,6378,6408,6437,%
8390   6467,6497,6526,6556,6585,6615,6644,6673,6703,6732,6762,6791,%
8391   6821,6851,6881,6910,6940,6969,6999,7028,7057,7087,7116,7146,%
8392   7175,7205,7235,7264,7294,7324,7353,7383,7412,7441,7471,7500,%
8393   7529,7559,7589,7618,7648,7678,7708,7737,7767,7796,7825,7855,%
8394   7884,7913,7943,7972,8002,8032,8062,8092,8121,8151,8180,8209,%
8395   8239,8268,8297,8327,8356,8386,8416,8446,8475,8505,8534,8564,%
8396   8593,8623,8652,8681,8711,8740,8770,8800,8829,8859,8889,8918,%
8397   8948,8977,9007,9036,9066,9095,9124,9154,9183,9213,9243,9272,%
8398   9302,9331,9361,9391,9420,9450,9479,9508,9538,9567,9597,9626,%
8399   9656,9686,9715,9745,9775,9804,9834,9863,9893,9922,9951,9981,%
8400   10010,10040,10069,10099,10129,10158,10188,10218,10247,10277,%
8401   10306,10335,10365,10394,10423,10453,10483,10512,10542,10572,%
8402   10602,10631,10661,10690,10719,10749,10778,10807,10837,10866,%
8403   10896,10926,10956,10986,11015,11045,11074,11103}
8404 \ExplSyntaxOff
8405 \</ca-chinese>

```



```

8427 \def\loadlocalcfg#1{%
8428   \openin0#1.cfg
8429   \ifeof0
8430     \closein0
8431   \else
8432     \closein0
8433     {\immediate\writel6{*****}%
8434      \immediate\writel6{* Local config file #1.cfg used}%
8435      \immediate\writel6{*}%
8436     }
8437     \input #1.cfg\relax
8438   \fi
8439   \@endofldef}

```

14.3 General tools

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

```

8440 \long\def\@firstofone#1{#1}
8441 \long\def\@firstoftwo#1#2{#1}
8442 \long\def\@secondoftwo#1#2{#2}
8443 \def\@nnil{\@nil}
8444 \def\@gobbletwo#1#2{}
8445 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8446 \def\@star@or@long#1{%
8447   \@ifstar
8448   {\let\l@ngrel@x\relax#1}%
8449   {\let\l@ngrel@x\long#1}}
8450 \let\l@ngrel@x\relax
8451 \def\@car#1#2\@nil{#1}
8452 \def\@cdr#1#2\@nil{#2}
8453 \let\@typeset@protect\relax
8454 \let\protected@edef\edef
8455 \long\def\@gobble#1{}
8456 \edef\@backslashchar{\expandafter\@gobble\string\\}
8457 \def\strip@prefix#1>{}
8458 \def\g@addto@macro#1#2{%
8459   \toks@{\expandafter{#1#2}%
8460   \xdef#1{\the\toks@}}
8461 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8462 \def\@nameuse#1{\csname #1\endcsname}
8463 \def\@ifundefined#1{%
8464   \expandafter\ifx\csname#1\endcsname\relax
8465     \expandafter\@firstoftwo
8466   \else
8467     \expandafter\@secondoftwo
8468   \fi}
8469 \def\@expandtwoargs#1#2#3{%
8470   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8471 \def\zap@space#1 #2{%
8472   #1%
8473   \ifx#2\@empty\else\expandafter\zap@space\fi
8474   #2}
8475 \let\bbl@trace\@gobble
8476 \def\bbl@error#1#2{%
8477   \begingroup
8478     \newlinechar=`^^J
8479     \def\{^^J(babel) }%
8480     \errhelp{#2}\errmessage{\{#1}%
8481   \endgroup}
8482 \def\bbl@warning#1{%
8483   \begingroup
8484     \newlinechar=`^^J
8485     \def\{^^J(babel) }%

```



```

8486 \message{\#1}%
8487 \endgroup}
8488 \let\bbl@infowarn\bbl@warning
8489 \def\bbl@info#1{%
8490 \begingroup
8491 \newlinechar=`^^J
8492 \def\{^J}%
8493 \wlog{#1}%
8494 \endgroup}

```

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

```

8495 \ifx\@preamblecmds\undefined
8496 \def\@preamblecmds{}
8497 \fi
8498 \def\@onlypreamble#1{%
8499 \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8500 \@preamblecmds\do#1}%
8501 \@onlypreamble\@onlypreamble

```

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

```

8502 \def\begindocument{%
8503 \@begindocumenthook
8504 \global\let\@begindocumenthook\undefined
8505 \def\do##1{\global\let##1\undefined}%
8506 \@preamblecmds
8507 \global\let\do\noexpand}

8508 \ifx\@begindocumenthook\undefined
8509 \def\@begindocumenthook{}
8510 \fi
8511 \@onlypreamble\@begindocumenthook
8512 \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`.

```

8513 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8514 \@onlypreamble\AtEndOfPackage
8515 \def\@endofldf{}
8516 \@onlypreamble\@endofldf
8517 \let\bbl@afterlang\@empty
8518 \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.

```

8519 \catcode`\&=\z@
8520 \ifx&\if@files\undefined
8521 \expandafter\let\csname if@files\expandafter\endcsname
8522 \csname iffalse\endcsname
8523 \fi
8524 \catcode`\&=4

```

Mimick \LaTeX 's commands to define control sequences.

```

8525 \def\newcommand{\@star@or@long\new@command}
8526 \def\new@command#1{%
8527 \@testopt{\@newcommand#1}0}
8528 \def\@newcommand#1[#2]{%
8529 \ifnextchar [{\@xargdef#1[#2]}%
8530 {\@argdef#1[#2]}}
8531 \long\def\@argdef#1[#2]#3{%
8532 \@yargdef#1\@ne{#2}{#3}}
8533 \long\def\@xargdef#1[#2][#3]#4{%
8534 \expandafter\def\expandafter#1\expandafter{%

```

```

8535 \expandafter\@protected@testopt\expandafter #1%
8536 \csname\string#1\expandafter\endcsname{#3}}%
8537 \expandafter\@yargdef \csname\string#1\endcsname
8538 \tw@{#2}{#4}}
8539 \long\def\@yargdef#1#2#3{%
8540 \@tempcnta#3\relax
8541 \advance \@tempcnta \@ne
8542 \let\@hash@\relax
8543 \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8544 \@tempcntb #2%
8545 \@whilenum\@tempcntb <\@tempcnta
8546 \do{%
8547 \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8548 \advance\@tempcntb \@ne}%
8549 \let\@hash@##%
8550 \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8551 \def\providecommand{\@star@or@long\providecommand}
8552 \def\providecommand#1{%
8553 \begingroup
8554 \escapechar\m@ne\xdef\@gtempa{\string#1}}%
8555 \endgroup
8556 \expandafter\@ifundefined\@gtempa
8557 {\def\reserved@a{\newcommand#1}}%
8558 {\let\reserved@a\relax
8559 \def\reserved@a{\newcommand\reserved@a}}%
8560 \reserved@a}%
8561 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8562 \def\declare@robustcommand#1{%
8563 \edef\reserved@a{\string#1}%
8564 \def\reserved@b{#1}%
8565 \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8566 \edef#1{%
8567 \ifx\reserved@a\reserved@b
8568 \noexpand\x@protect
8569 \noexpand#1%
8570 \fi
8571 \noexpand\protect
8572 \expandafter\noexpand\csname
8573 \expandafter\@gobble\string#1 \endcsname
8574 }%
8575 \expandafter\newcommand\csname
8576 \expandafter\@gobble\string#1 \endcsname
8577 }
8578 \def\x@protect#1{%
8579 \ifx\protect\@typeset@protect\else
8580 \@x@protect#1%
8581 \fi
8582 }
8583 \catcode\&=\z@ % Trick to hide conditionals
8584 \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`.

```

8585 \def\bbl@tempa{\csname newif\endcsname&\ifin@}
8586 \catcode\&=4
8587 \ifx\in@\@undefined
8588 \def\in@#1#2{%
8589 \def\in@##1#2##3\in@{%
8590 \ifx\in@##2\in@false\else\in@true\fi}%
8591 \in@##1\in@\in@@}
8592 \else
8593 \let\bbl@tempa\empty

```

```
8594 \fi
8595 \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).

```
8596 \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.

```
8597 \def\@ifl@aded#1#2#3#4{}
```

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

```
8598 \ifx\@tempcnta\undefined
8599   \csname newcount\endcsname\@tempcnta\relax
8600 \fi
8601 \ifx\@tempcntb\undefined
8602   \csname newcount\endcsname\@tempcntb\relax
8603 \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`).

```
8604 \ifx\bye\undefined
8605   \advance\count10 by -2\relax
8606 \fi
8607 \ifx\@ifnextchar\undefined
8608   \def\@ifnextchar#1#2#3{%
8609     \let\reserved@#1%
8610     \def\reserved@a{#2}\def\reserved@b{#3}%
8611     \futurelet\@let@token\@ifnch}
8612   \def\@ifnch{%
8613     \ifx\@let@token\@sptoken
8614       \let\reserved@c\@xifnch
8615     \else
8616       \ifx\@let@token\reserved@d
8617         \let\reserved@c\reserved@a
8618       \else
8619         \let\reserved@c\reserved@b
8620       \fi
8621     \fi
8622     \reserved@c}
8623   \def\:{\let\@sptoken= }\: % this makes \@sptoken a space token
8624   \def\:{\@xifnch} \expandafter\def\:{ \futurelet\@let@token\@ifnch}
8625 \fi
8626 \def\@testopt#1#2{%
8627   \@ifnextchar[#{#1}{#1[#2]}}
8628 \def\@protected@testopt#1{%
8629   \ifx\protect\@typeset@protect
8630     \expandafter\@testopt
8631   \else
8632     \@x@protect#1%
8633   \fi}
8634 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8635   #2\relax}\fi}
8636 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8637   \else\expandafter\@gobble\fi{#1}}
```

14.4 Encoding related macros

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

```

8638 \def\DeclareTextCommand{%
8639   \@dec@text@cmd\providecommand
8640 }
8641 \def\ProvideTextCommand{%
8642   \@dec@text@cmd\providecommand
8643 }
8644 \def\DeclareTextSymbol#1#2#3{%
8645   \@dec@text@cmd\chardef#1{#2}#3\relax
8646 }
8647 \def\@dec@text@cmd#1#2#3{%
8648   \expandafter\def\expandafter#2%
8649     \expandafter{%
8650       \csname#3-cmd\expandafter\endcsname
8651       \expandafter#2%
8652       \csname#3\string#2\endcsname
8653     }%
8654 %   \let\@ifdefinable\@rc@ifdefinable
8655   \expandafter#1\csname#3\string#2\endcsname
8656 }
8657 \def\@current@cmd#1{%
8658   \ifx\protect\@typeset@protect\else
8659     \noexpand#1\expandafter\@gobble
8660   \fi
8661 }
8662 \def\@changed@cmd#1#2{%
8663   \ifx\protect\@typeset@protect
8664     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8665       \expandafter\ifx\csname ?\string#1\endcsname\relax
8666         \expandafter\def\csname ?\string#1\endcsname{%
8667           \@changed@x@err{#1}%
8668         }%
8669       \fi
8670       \global\expandafter\let
8671         \csname\cf@encoding \string#1\expandafter\endcsname
8672         \csname ?\string#1\endcsname
8673     \fi
8674     \csname\cf@encoding\string#1%
8675       \expandafter\endcsname
8676   \else
8677     \noexpand#1%
8678   \fi
8679 }
8680 \def\@changed@x@err#1{%
8681   \errhelp{Your command will be ignored, type <return> to proceed}%
8682   \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8683 \def\DeclareTextCommandDefault#1{%
8684   \DeclareTextCommand#1?%
8685 }
8686 \def\ProvideTextCommandDefault#1{%
8687   \ProvideTextCommand#1?%
8688 }
8689 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8690 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8691 \def\DeclareTextAccent#1#2#3{%
8692   \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
8693 }
8694 \def\DeclareTextCompositeCommand#1#2#3#4{%
8695   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8696   \edef\reserved@b{\string##1}%
8697   \edef\reserved@c{%
8698     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8699   \ifx\reserved@b\reserved@c
8700     \expandafter\expandafter\expandafter\ifx

```

```

8701 \expandafter\@car\reserved@a\relax\relax\@nil
8702 \@text@composite
8703 \else
8704 \edef\reserved@b##1{%
8705 \def\expandafter\noexpand
8706 \csname#2\string#1\endcsname###1{%
8707 \noexpand\@text@composite
8708 \expandafter\noexpand\csname#2\string#1\endcsname
8709 ###1\noexpand\@empty\noexpand\@text@composite
8710 {##1}%
8711 }%
8712 }%
8713 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8714 \fi
8715 \expandafter\def\csname\expandafter\string\csname
8716 #2\endcsname\string#1-\string#3\endcsname{#4}
8717 \else
8718 \errhelp{Your command will be ignored, type <return> to proceed}%
8719 \errmessage{\string\DeclareTextCompositeCommand\space used on
8720 inappropriate command \protect#1}
8721 \fi
8722 }
8723 \def\@text@composite#1#2#3\@text@composite{%
8724 \expandafter\@text@composite@x
8725 \csname\string#1-\string#2\endcsname
8726 }
8727 \def\@text@composite@x#1#2{%
8728 \ifx#1\relax
8729 #2%
8730 \else
8731 #1%
8732 \fi
8733 }
8734 %
8735 \def\@strip@args#1:#2-#3\@strip@args{#2}
8736 \def\DeclareTextComposite#1#2#3#4{%
8737 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8738 \bgroup
8739 \lccode`\@=#4%
8740 \lowercase{%
8741 \egroup
8742 \reserved@a @%
8743 }%
8744 }
8745 %
8746 \def\UseTextSymbol#1#2{#2}
8747 \def\UseTextAccent#1#2#3{}
8748 \def\@use@text@encoding#1{}
8749 \def\DeclareTextSymbolDefault#1#2{%
8750 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8751 }
8752 \def\DeclareTextAccentDefault#1#2{%
8753 \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8754 }
8755 \def\cf@encoding{OT1}

```

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

```

8756 \DeclareTextAccent{"}{OT1}{127}
8757 \DeclareTextAccent{'}{OT1}{19}
8758 \DeclareTextAccent{^}{OT1}{94}
8759 \DeclareTextAccent{\`}{OT1}{18}
8760 \DeclareTextAccent{\~}{OT1}{126}

```

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

```
8761 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8762 \DeclareTextSymbol{\textquotedblright}{OT1}{`\'}
8763 \DeclareTextSymbol{\textquoteleft}{OT1}{`\'}
8764 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8765 \DeclareTextSymbol{\i}{OT1}{16}
8766 \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`.

```
8767 \ifx\scriptsize\@undefined
8768   \let\scriptsize\sevenrm
8769 \fi
```

And a few more “dummy” definitions.

```
8770 \def\language{english}%
8771 \let\bbl@opt@shorthands\@nnil
8772 \def\bbl@ifshorthand#1#2#3{#2}%
8773 \let\bbl@language@opts\@empty
8774 \let\bbl@ensureinfo\@gobble
8775 \let\bbl@provide@locale\relax
8776 \ifx\babeloptionstrings\@undefined
8777   \let\bbl@opt@strings\@nnil
8778 \else
8779   \let\bbl@opt@strings\babeloptionstrings
8780 \fi
8781 \def\BabelStringsDefault{generic}
8782 \def\bbl@tempa{normal}
8783 \ifx\babeloptionmath\bbl@tempa
8784   \def\bbl@mathnormal{\noexpand\textormath}
8785 \fi
8786 \def\AfterBabelLanguage#1#2{}
8787 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8788 \let\bbl@afterlang\relax
8789 \def\bbl@opt@safe{BR}
8790 \ifx\uclclist\@undefined\let\uclclist\@empty\fi
8791 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8792 \expandafter\newif\csname ifbbl@single\endcsname
8793 \chardef\bbl@bidimode\z@
8794 </Emulate LaTeX>
```

A proxy file:

```
8795 <*\plain>
8796 \input babel.def
8797 </\plain>
```

15 Acknowledgements

I would like to thank all who volunteered as β -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful. There are also many contributors for specific languages, which are mentioned in the respective files. Without them, babel just wouldn't exist.

References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national \TeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

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